

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2009

A new approach in building parallel finite field multipliers

Mohammadali Sharifan

University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Sharifan, Mohammadali, "A new approach in building parallel finite field multipliers" (2009). *Electronic Theses and Dissertations*. 8128.

<https://scholar.uwindsor.ca/etd/8128>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

A NEW APPROACH IN BUILDING PARALLEL FINITE FIELD MULTIPLIERS

by

Mohammadali Sharifan

A Thesis

Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2009

© 2009 Mohammadali Sharifan



Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-57623-6
Our file Notre référence
ISBN: 978-0-494-57623-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■+■
Canada

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

A new method for building bit-parallel polynomial basis finite field multipliers is proposed in this thesis. Among the different approaches to build such multipliers, Mastrovito multipliers based on a trinomial, an all-one-polynomial, or an equally-spaced-polynomial have the lowest complexities. The next best in this category is a conventional multiplier based on a pentanomial. Any newly presented method should have complexity results which are at least better than those of a pentanomial based multiplier. By applying our method to certain classes of finite fields we have gained a space complexity as $n^2 + n - 4$ and a time complexity as $T_A + (\lceil \log_2(n-1) \rceil + 3)T_X$ which are better than the lowest space and time complexities of a pentanomial based multiplier found in literature. Therefore this multiplier can serve as an alternative in those finite fields in which no trinomial, all-one-polynomial or equally-spaced-polynomial exists.

To all the things that I lost, while trying to gain.

Acknowledgments

I would like to thank Dr. Huapeng Wu for all the help and support that he gave me as my supervisor, and for always encouraging me through the hard times that I had during this research project.

I also want to thank Dr. Henry Hu for precisely reviewing my work and Dr. Sazzadur Chowdhury for his valuable advice and comments.

I cannot be grateful enough to my parents for all the sacrifices they made to give me whatever I need to gain my goals.

Special thanks to S. for her endless love, support, and patience.

Table of Contents

Abstract	iv
Dedication	v
Acknowledgments	vi
List of Tables	ix
List of Figures	x
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Outline	3
2 Mathematical Background	5
2.1 Fundamental Concepts	5
2.2 Modular Arithmetic	8
2.3 Finite Fields	11
2.4 Polynomials and Finite Fields	12
2.5 Extension Finite Fields	16
2.6 Basis in Finite Fields	19
2.7 Finite Field Multiplication in Polynomial Basis	20
2.8 Multiplier Architecture and Complexity	24
3 An Overview of Finite Field Multipliers	26
3.1 Parallel Finite Field Multipliers	27
3.2 Serial Finite Field Multipliers	50
3.3 Summary	54

4	New Finite Field Multiplier	57
4.1	Introduction	57
4.2	Multiplier Architecture.....	59
4.3	Applying the New Method to Classes of Finite Fields	68
4.3.1	Review of Three Classes of Finite Fields	69
4.3.2	Results Related to Type I Polynomials.....	75
4.3.3	Results Related to Type II Polynomials.....	78
4.3.4	Results Related to Type III Polynomials	80
4.4	Comparisons.....	81
5	Conclusion	82
5.1	Summary of contributions.....	82
5.2	Future Work	83
	References.....	84
	VITA AUCTORIS	89

List of Tables

Table 2.1 Addition in $GF(2^3)$	18
Table 2.2 Multiplication in $GF(2^3)$	18
Table 2.3 Multiplicative Inverse Elements in $GF(2^3)$	19
Table 3.1 Finite Field Multipliers Based on Hardware Architecture.....	54
Table 3.2 Finite Field Multipliers Based on Element Representation	55
Table 3.3 Best Complexities of Different Finite Field Multipliers.....	56
Table 4.1 Complexity Results of a Type I Multiplier	78
Table 4.2 Complexity Results of a Type II Multiplier.....	79
Table 4.3 Complexity Results of a Type III Multiplier	81
Table 4.4 The Results of the New Multiplier vs the Best Pentanomial Multiplier.....	81

List of Figures

Figure 2.1 Arithmetic Operations in Z_6	10
Figure 2.2 Arithmetic Operations in $GF(5)$	12
Figure 3.1 Reduction Array for a General Trinomial	36
Figure 3.2 Reduction Array for $x^n + x^{m+1} + x^m + x + 1$	43
Figure 3.3 A 5×5 Toeplitz Matrix.....	49
Figure 3.4 Berlekamp Multiplier	51
Figure 3.5 Massey-Omura Serial Multiplier	54
Figure 4.1 V Matrices of $GF(2^5)$ with $f(x) = x^5 + x^2 + 1$	65
Figure 4.2 Proposed Multiplier Architecture	67
Figure 4.3 Transfer Matrix of a Type I Polynomial.....	71
Figure 4.4 T Matrix of a Type II Polynomial	72
Figure 4.5 T Matrix of a Type III Polynomial.....	75
Figure 4.6 Longest Signal Pass of a Type I Multiplier	78

1 Introduction

1.1 Motivation

In the current age of information technology, while the internet is growing very fast in almost every aspect of life; electronic communication of very private and important data is a common task. Consequently, network security has become a primary demand for IT service providers and users. No matter what the application is, from sharing family pictures in a social network to sending important customer lists through a business email or checking your bank account balance online, information transaction should be done with a very high level of confidentiality. Secure transactions are even more vital when it comes to e-commerce, online money transactions or governmental information transactions such as social insurance numbers or tax returns. Almost all required network security services can be achieved by *Cryptography* [35].

Encryption is one common cryptography algorithm which means scrambling the data in a way that only authorized users with proper authorization (a necessary key) can unscramble it. Cryptography algorithms fall into two main categories: *symmetric key* and *public key*.

Symmetric key cryptosystems rely on a key distribution center which provides a special and unique key to each user in each pair of communication. In this method the number of

required keys for a network grows very fast by increasing the number of users. Therefore, in large systems symmetric key cryptosystems encounter key management and key distribution problems. This problem led to the introduction of the *public key* cryptography by Diffie and Hellman in 1976 [3].

In a public key network, each user is assigned two special keys. One key is kept private while the other is known to all the other users in the network. Any message addressed to the user is scrambled with the user's public key. However, in order to unscramble this data, only the user's private key is applicable. Different methods have been introduced for public key. In 1978 Rivest, Shamir and Adleman [33] introduced the so-called RSA algorithm which is by far the most popular public key algorithm. About a decade later in 1985 El-Gamal [4] introduced another technique in cryptography named after himself. At the same time Koblitz [16] and Miller [24] independently introduced the *elliptic curve cryptography* (ECC) where finite fields came into play. This method is based on the group of points on an elliptic curve (EC) over a finite field.

Despite the fact that the vast majority of the products and standards that use public key cryptography for encryption use the RSA method, ECC is more efficient compared to RSA. This is because ECC can offer equal security with a far smaller key size, hence reducing processing overhead. Since the RSA standard key length has recently greatly increased, the issue of the key length has come more into attention, especially that the increase in key length has put a burden on e-commerce sites that conduct large number of secure transactions. This is leading to a vast replacement of RSA by ECC in many different security products in hardware and software.

Elliptic curve cryptography (ECC) is based on the elliptic curve discrete logarithm problem [10]. *EC* calculation over a finite field is based on finite field addition, subtraction, multiplication, squaring and division amongst which multiplication is the most important operation to implement.

Although implementing cryptography algorithms in the software level is much easier, they are considerably slower compared to hardware implementations. This will effectively slow down its processing and increase the consumption of the valuable time of the main processor of the hosting system. On the other hand, the hardware implementation of a cryptosystem will result in faster processors and the opportunity to have larger key lengths. The advantages of these characteristics can be clearly seen where there is a large volume of secure transactions. Hence hardware implemented crypto-processors result in a higher level of security with a better performance.

Efficient hardware implementation of ECC systems highly depends on the efficiency of finite field multiplication. This is due to the fact that when calculating the speed of an elliptic curve processor, finite field multiplication is considered to be the most time consuming operation. Finite field addition (and in some systems squaring) is considered to be almost free compared to multiplication [5]. The importance of improving finite field multiplication algorithms provides the main motivation of this research.

In this thesis a new efficient parallel finite field multiplier is proposed which can be used as an alternative for present methods of multiplication in hardware implementation of elliptic curve cryptosystems.

Application of finite field multipliers is not restricted to elliptic curve cryptography. Before the introduction of ECC, finite fields had come into attention in coding theory and error detecting codes especially in Reed-Solomon encoders [30]. Finite field arithmetic is also used in combinatorial designs [38], and computer algebra [19]. This shows the extensive application of finite field multiplication algorithms. Nevertheless this thesis mainly focuses on the application of finite field in public key cryptography.

1.2 Thesis Outline

Chapter 2 gives an introduction to finite fields and the arithmetic within these fields. It briefly covers the mathematical background of the construction of finite fields and

explains different methods of multiplication. Various classes of finite fields are introduced and different methods of representing elements in finite fields are reviewed.

Chapter 3 provides a comprehensive survey on the state-of-art technologies on building finite field multipliers. It explains different types of implementations of finite field multipliers and compares pros and cons of each method. It also provides different types of fields used in finite field multipliers and since this thesis proposes a parallel polynomial basis finite field multiplier all the similar existing multipliers are provided with their exact space and speed complexities.

Chapter 4 introduces the new method of finite field multiplication. It provides complete hardware architecture for this multiplier. Also some recently introduced types of finite fields are reviewed and the new architecture of multiplier is applied to these fields. Finally it provides the results of this multiplier for those fields in terms of space and time complexity of the hardware.

In the end, Chapter 5 presents the concluding remarks surrounding our proposed method, and suggests future work on this subject.

2 Mathematical Background

2.1 Fundamental Concepts

Groups, rings and fields are the fundamental elements of a branch of mathematics called *abstract algebra*. In this branch we deal with certain sets and their elements on which we can operate algebraically; meaning that by combining two elements of the set in several ways a third element of the set can be obtained. There are certain rules which control these operations. It is important to note that although the notations of these operations are usually similar to addition or multiplication; in abstract algebra we are not limited to ordinary arithmetical operations.

Groups

Definition 2.1 A group G ; here denoted by $\{g, *\}$, is a set G together with a binary operation $*$ on G such that the following properties hold:

1. *Closure*; that is, for all a, b in G , $a * b$ is in G
2. *Associativity*; that is, for all a, b, c in G , $a * (b * c) = (a * b) * c$
3. *Unity element*; there is an element e in G such that for all a in G , $a * e = e * a = a$
4. *Inverse element*; for all a in G there exists a^{-1} in G such that $a * a^{-1} = a^{-1} * a = e$

It is important to note that “*” is not necessarily “×” although it could be “×” or “+”.

Commutative Group

If the group also satisfies

5. For all a, b in G , $a * b = b * a$

Then the group is called *commutative* or *abelian*.

Finite Group

If the group contains finite number of elements then it is called a *finite* group. The *order* of a group is the number of elements in the group.

Cyclic Group

Exponentiation in the group is defined as repeated application of the group operator. Here if a fixed element a exists in G in a way that every element in G is a power a^k , then the group is called *cyclic* and element a is called the *generator of the group*.

If the operation of the group is “+” then the group is called *additive* and the unity element is represented as “0”. If the operation of the group is “×”, then the group is called *multiplicative* and the unity element is represented by “1”.

Rings

Definition 2.2 A ring R ; here denoted by $\{R, +, \times\}$, is a set R and two binary operations addition and multiplication in a way that the following properties are satisfied:

1. R is a commutative group with respect to “+”

2. Closure for “ \times ”; that is, for all a, b in R , $a \times b$ is also in R
3. Associativity of “ \times ”; that is, for all a, b, c in R , $a \times (b \times c) = (a \times b) \times c$
4. Distributive laws for “ \times ” over “ $+$ ”; that is, for all a, b, c in R ,

$$a \times (b + c) = a \times b + a \times c \text{ and } (a + b) \times c = a \times c + b \times c$$

Commutative Ring

If the ring also satisfies

5. For all a, b in R , $a \times b = b \times a$

Then the ring is considered *commutative*.

Integral Domain

If a commutative ring satisfies the following axioms:

6. *Unity element* for “ \times ”; there is an element “1” in R such that for all a in R ,

$$a \times 1 = 1 \times a = a$$
7. *No zero divisors*; if a, b in R and $a \times b = 0$ then either $a = 0$ or $b = 0$

Then it is called an *integral domain*.

Fields

Definition 2.3 A field F , which we denote by $\{F, +, \times\}$, is a set of elements with two binary operations addition and multiplication such that the following properties are hold:

1. F is an integral domain
2. *Multiplicative inverse*; for all a in F there is an element a^{-1} in F such that

$$a \times a^{-1} = a^{-1} \times a = 1; \text{ except for “0”}.$$

In other words, a field is a set in which addition, subtraction, multiplication and division is possible without leaving the set. Division is defined as $a / b = a \times (b^{-1})$ and subtraction is defined as $a - b = a + (-b)$ in which $-b$ is defined as additive inverse of b .

Finite Field

If the number of elements of the field is finite then it is called a *finite field*.

2.2 Modular Arithmetic

Dividing any integer a by any positive integer n will result in an integer quotient q and a remainder r , which is a positive integer. This relationship is demonstrated as:

$$a = qn + r \quad \text{where} \quad 0 \leq r < n \quad \text{and} \quad q = \lfloor a/n \rfloor \quad (2.1)$$

In which $\lfloor x \rfloor$ is the largest integer less than or equal to x . The remainder r is often referred to as *residue*.

“mod” Operator

For an integer a , and a positive integer n ; $a \bmod n$ is defined as the residue of a / n .

Congruent Modulo

Two integers a, b are congruent modulo n if $a \bmod n = b \bmod n$ and it is written as:

$$a \equiv b \bmod n.$$

Divisor

If $a \bmod b = 0$ then b is called a *divisor* of a and it is written as: $b|a$.

Residue Class

A Residue class is the set of all integers which are congruent modulo n . Each class is represented by the smallest non-negative integer in the class. Consider a set Z_n which contains all non-negative integers less than n . This set contains all the residue classes modulo n . For example $Z_3 = \{0, 1, 2\}$ shows all residue classes of $n = 3$. These classes are:

$$[0] = \{\dots, -9, -6, -3, 0, 3, 6, 9, \dots\}$$

$$[1] = \{\dots, -8, -5, -2, 1, 4, 7, 10, \dots\}$$

$$[2] = \{\dots, -7, -4, -1, 2, 5, 8, 11, \dots\}$$

Reducing modulo n

For each integer k , finding the smallest non-negative integer to which k is congruent modulo n is called *reduction modulo n* .

By definition, the “mod n ” operator maps all integers into Z_n . This suggests the idea of performing arithmetic operations within the confines of this set. This type of arithmetic is called *modular arithmetic* and it has the following basic properties:

1. $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
2. $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

If we perform this modular arithmetic only on integers in Z_n , with the following properties, it can be seen that Z_n is a commutative ring with an identity element for multiplication:

1. Commutative laws:
 - i. $(a + b) \bmod n = (b + a) \bmod n$
 - ii. $(a \times b) \bmod n = (b \times a) \bmod n$

2. Associative laws:

- i. $[(a + b) + c] \bmod n = [a + (b + c)] \bmod n$
- ii. $[(a \times b) \times c] \bmod n = [a \times (b \times c)] \bmod n$

3. Distributive laws:

- i. $[a \times (b + c)] \bmod n = [(a \times b) + (a \times c)] \bmod n$
- ii. $[a + (b \times c)] \bmod n = [(a + b) \times (a + c)] \bmod n$

4. Unity elements:

- i. $(0 + a) \bmod n = (a + 0) \bmod n = a \bmod n$
- ii. $(1 \times a) \bmod n = (a \times 1) \bmod n = a \bmod n$

5. Additive inverse:

- i. For each a in Z_n there exists b in Z_n such that $a + b \equiv 0 \bmod n$

An example would clarify this better. Considering Z_6 , the elements of this field are: $\{0, 1, 2, 3, 4, 5\}$. Figure (2.1) shows the arithmetic operations in Z_6 :

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1		2	3	4	5	0
2			4	5	0	1
3				0	1	2
4					2	3
5						4

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1		1	2	3	4	5
2			4	0	2	4
3				3	0	3
4					4	2
5						1

a	-a	a ⁻¹
0	0	—
1	5	1
2	4	—
3	3	—
4	2	—
5	1	5

Figure 2.1 Arithmetic Operations in Z_6

It can be seen that multiplicative inverse doesn't exist for all elements of Z_n . The reason behind that comes from a concept called *relatively prime*. Two integers are considered to be *relatively prime* (*co-prime*) if their only common positive integer factor is 1. For all a in Z_n , a has a multiplicative inverse in Z_n if and only if a and n are relatively prime. In the

above example 1 and 5 had a multiplicative inverse in Z_6 but 2, 3 and 4 didn't because they have common factors other than 1 with 6 which are 2, 3 and 2 respectively.

It can be concluded from the above concept that in order to have a multiplicative inverse for all elements of Z_n , n should be co-prime to all numbers in the set. This leads to the conclusion that n should be a *prime number*. So for a prime number p , all elements of Z_p have a multiplicative inverse within the set. This is the basic concept of finite fields.

2.3 Finite Fields

In early 19th century, French mathematician Évariste Galois introduced finite fields, however they did not come into attention until 1960's when coding methods were developed. But the introduction of elliptic curve cryptography in 1985 made finite fields very popular. Finite fields play a crucial role in many cryptographic algorithms especially in ECC.

It can be shown that the order of a finite field must be a power of a prime p^n , where p is a prime number and n is a positive integer. The finite field of the order p^n is usually denoted as $GF(p^n)$; GF stands for Galois field in honour of the French mathematician. There are two different types of finite fields:

- Ground Fields, $GF(p)$; where $n = 1$
- Extension Fields, $GF(p^n)$

We will talk about the ground fields in this section. Extension fields will be discussed in section 2.5.

As mentioned before a set Z_n together with modular operations is a commutative ring. We also saw that if n is a prime number then all the non-zero elements in the set Z_n have a multiplicative inverse within the set. So according to the definitions in section 2.1, set Z_p

together with modular arithmetic operations makes a field. Since the number of elements in this field is finite, Z_p is a finite field:

$$GF(p) = Z_p \tag{2.2}$$

As an example, arithmetic operations $GF(5)$ are depicted in figure (2.2):

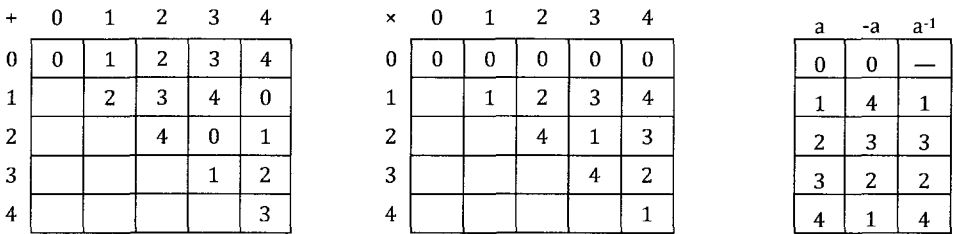


Figure 2.2 Arithmetic Operations in GF(5)

The simplest finite field is $GF(2)$. This field has particular importance since it is used to built extension fields $GF(2^n)$, with n as a positive integer. Extension fields of form $GF(2^n)$ are very important in implementing cryptographic algorithms in computer hardware. Arithmetic operations in field GF(2) are summarized as follows:

+	0	1	×	0	1	a	-a	a ⁻¹
0	0	1	0	0	0	0	0	—
1	1	0	1	0	1	1	1	1

It can be seen that addition in this field is equivalent to the XOR operation, and multiplication is equivalent to the AND operation. Another important property is that since the additive inverse of each element in this field is the element itself, subtraction is equivalent to addition. In other words for all a, b in $GF(2)$, $a - b = a + (-b) = a + b$.

2.4 Polynomials and Finite Fields

In elementary algebra a polynomial is regarded as an expression of the form:

$$f(x) = \sum_{i=0}^n a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$$

Here, n is the degree of the polynomial. a_i s are coefficients, and x is viewed as a variable. Since we are usually not interested in evaluating a polynomial for a particular value of x in abstract algebra, we refer to it as the *indeterminate*.

Constant Polynomial

When $n = 0$, the polynomial is called a constant polynomial which is actually only a coefficient a_0 .

Monic Polynomial

For a polynomial of degree n , when $a_n = 1$, the polynomial is called a *monic* polynomial.

In general, coefficients of a polynomial belong to a specific set. If this set is a ring, then it can be shown that the polynomials over this ring form a *polynomial ring*. Basic polynomial arithmetic includes the operations of addition, subtraction and multiplication. Addition in this polynomial ring is defined as:

$$\begin{aligned} A(x) &= \sum_{i=0}^n a_i x^i \quad \text{and} \quad B(x) = \sum_{i=0}^n b_i x^i \\ C(x) &= A(x) + B(x) = \sum_{i=0}^n c_i x^i = \sum_{i=0}^n (a_i + b_i) x^i \end{aligned} \tag{2.3}$$

Note that even if $A(x)$ and $B(x)$ are not of the same degree, this formulation is correct considering that the polynomial with the smaller degree would be filled with 0 coefficients.

Likewise, multiplication in this ring is defined as:

$$\begin{aligned}
A(x) &= \sum_{i=0}^n a_i x^i \quad \text{and} \quad B(x) = \sum_{i=0}^m b_i x^i \\
C(x) &= A(x) \times B(x) = \sum_{k=0}^{n+m} c_k x^k
\end{aligned} \tag{2.4}$$

where

$$c_k = \sum_{\substack{i+j=k \\ 0 \leq i \leq n; 0 \leq j \leq m}} a_i b_j \tag{2.5}$$

Subtraction in this ring is defined like addition using the additive inverse of the coefficients. Division can also be defined in this polynomial ring if the set of coefficients is a field instead of a ring. Note that this division will result in both quotient and remainder, and since there isn't a multiplicative inverse for any polynomial in this ring, these polynomials do not form a field:

$$\begin{aligned}
A(x) &= \sum_{i=0}^n a_i x^i \quad \text{and} \quad B(x) = \sum_{i=0}^m b_i x^i \\
\frac{A(x)}{B(x)} &= q(x) + \frac{r(x)}{B(x)}
\end{aligned} \tag{2.6}$$

Considering $n \geq m$, the degree of $q(x)$ is $m - n$, and the degree of $r(x)$ is less than or equal to $m - 1$.

Polynomials over Z_p

As mentioned in the previous section set Z_p forms a finite field, so division can be defined for polynomials over Z_p or generally speaking over prime finite fields. This is the basis of forming extension finite fields $GF(p^n)$. For cryptographic applications, polynomials over $GF(2)$ are of most interest. As we will show later these polynomials can be easily stored as binary numbers in computer memory. Also addition and multiplication on these

polynomials is easily implemented by logical XOR, and AND gates. Here is an example of these polynomials. Note that the coefficients here are either 0 or 1:

$$A(x) = x^5 + x^3 + x^2 + 1$$

$$B(x) = x^4 + x^2 + 1$$

Addition:

$$A(x) + B(x) = x^5 + x^4 + x^3 + x^2 + x^2 + 1 + 1 = x^5 + x^4 + x^3$$

Subtraction:

$$A(x) - B(x) = x^5 - x^4 + x^3 + x^2 - x^2 + 1 - 1 = x^5 + x^4 + x^3$$

Multiplication:

$$\begin{aligned} A(x) \times B(x) &= x^9 + x^7 + x^6 + x^4 + x^7 + x^5 + x^4 + x^2 + x^5 + x^3 + x^2 + 1 \\ &= x^9 + x^6 + x^5 + x^3 + 1 \end{aligned}$$

Division:

$$\frac{A(x)}{B(x)} = \frac{x^5 + x^3 + x^2 + 1}{x^4 + x^2 + 1} = x + \frac{x^2 + x + 1}{x^4 + x^2 + 1}$$

Irreducible Polynomial

A polynomial $P(x)$ over a field is considered irreducible if $P(x) = A(x) \times B(x)$ implies that either $A(x)$ or $B(x)$ is a constant polynomial. In other words $P(x)$ is irreducible if it cannot be expressed as a product of two other polynomials over the same field and only allows trivial factorizations. Irreducible polynomials are also called *prime* polynomials as an analogy to prime numbers.

Example: $P(x) = x^2 - 2$ over the field of rational numbers is irreducible but $P(x) = x^2 - 2 = (x + \sqrt{2})(x - \sqrt{2})$ over the field of real numbers is indeed reducible.

2.5 Extension Finite Fields

Let F be a field. A subset K of F which is a field under the operations of F is called a *subfield* of F . In this context, F is called an *extension field* of K . For example the set of complex numbers is an extension field for the set of real numbers. With the help of polynomial arithmetic and irreducible polynomials, extension fields exist for finite fields as well.

Earlier in this chapter, we mentioned that the order of a finite field must be of the form p^n , in which p is a prime and n is a positive integer. We found that when $n = 1$ we have a prime finite field which is the set Z_p and with all the operations performed modulo p , all the axioms for a field is satisfied. But as we know, set Z_{p^n} with operations modulo p^n does not make a field because, as we mentioned before, since p^n is not a prime number we cannot find a multiplicative inverse for all elements of the field.

Modular Polynomial Arithmetic

As mentioned before, the set of all polynomials over a field develops a ring which allows division within the set. Considering the set S of all polynomials of degree $n-1$ or less over the finite field $GF(p)$, each polynomial has the form

$$f(x) = \sum_{i=0}^n a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$$

Where

$$a_i \in \{0, 1, \dots, p-1\}$$

The total number of distinct polynomials within this set is p^n . Based on the following definition of arithmetic operations, set S becomes a finite field:

1. Arithmetic on coefficients is performed modulo p .
2. Arithmetic on polynomials is performed modulo some irreducible polynomial $P(x)$ of degree n .

It is important to note that this polynomial modular operation only happens when multiplying. When performing addition, the degree of the result will never exceed $n-1$ so there is no need for reducing modulo $P(x)$. However in the case of multiplying, the result can have a degree of at most $2n-2$. Hence, for the results that have a degree more than $n-1$, reduction is necessary.

It can be shown that the set of residues modulo $P(x)$ of degree n consists of p^n elements which are all polynomials of degree $n-1$ or less. So with analogy to integers we can write $S = Z_{P(x)}$. This set satisfies all the axioms of a finite field so as an extension finite field we have:

$$GF(p^n) = S = Z_{P(x)} \quad (2.7)$$

When implementing encryption algorithms in computer hardware, we usually deal with finite fields of form $GF(2^n)$. The reason is that any polynomial in this field can uniquely be represented with a binary number, because the coefficients of the polynomials in this field are either 0 or 1. Also each polynomial here represents an integer number within the range 0 to $2^n - 1$. From now on we only refer to finite fields of form $GF(2^n)$ because of their application in cryptographic hardware.

Consider $GF(2^3)$ with Irreducible polynomial $P(x) = x^3 + x^2 + 1$. This field has 8 elements which are:

0	000	0
1	001	1
x	010	2
x + 1	011	3
x ²	100	4
x ² + 1	101	5
x ² + x	110	6
x ² + x + 1	111	7

Addition and multiplication in this field is illustrated in the following tables:

			0	1	x	x + 1	x ²	x ² + 1	x ² + x	x ² + x + 1
			000	001	010	011	100	101	110	111
+			0	1	2	3	4	5	6	7
0	000	0	0	1	2	3	4	5	6	7
1	001	1		0	3	2	5	4	7	6
x	010	2			0	1	6	7	4	5
x + 1	011	3				0	7	6	5	4
x ²	100	4					0	1	2	3
x ² + 1	101	5						0	3	2
x ² + x	110	6							0	1
x ² + x + 1	111	7								0

Table 2.1 Addition in GF(2³)

			0	1	x	x + 1	x ²	x ² + 1	x ² + x	x ² + x + 1
			000	001	010	011	100	101	110	111
×			0	1	2	3	4	5	6	7
0	000	0	0	0	0	0	0	0	0	0
1	001	1		1	2	3	4	5	6	7
x	010	2			4	6	5	7	1	3
x + 1	011	3				5	1	2	7	4
x ²	100	4					7	3	2	6
x ² + 1	101	5						6	4	1
x ² + x	110	6							3	5
x ² + x + 1	111	7								2

Table 2.2 Multiplication in GF(2³)

A			a ⁻¹		
0	000	0	—	—	—
1	001	1	1	001	1
x	010	2	6	010	x
x + 1	011	3	4	011	x + 1
x ²	100	4	3	100	x ²
x ² + 1	101	5	7	101	x ² + 1
x ² + x	110	6	2	110	x ² + x
x ² + x + 1	111	7	5	111	x ² + x + 1

Table 2.3 Multiplicative Inverse Elements in GF(2³)

2.6 Basis in Finite Fields

In the previous example we represented field elements as a polynomial of degree less than or equal to 2. But there are other ways to represent the field elements. In general, any element a of a finite field $GF(2^n)$ can be represented as:

$$a = a_0\beta_0 + a_1\beta_1 + \cdots + a_{n-1}\beta_{n-1} \quad (2.8)$$

where $a_i \in GF(2)$ and $\beta_i \in GF(2^n)$ for $0 \leq i \leq n - 1$

The set $\{\beta_0, \beta_1, \dots, \beta_{n-1}\}$ which contains n linear independent elements is the *basis* of the field [23]. There are different types of basis for finite fields. Some of the most important types are:

- Polynomial Basis
- Normal Basis [31]
- Dual Basis [25]
- Triangular Basis [11]
- Weakly Dual Basis [39]
- Redundant Representation [40]
- Shifted Polynomial Basis [29]

In the previous example, elements of the finite field were represented in *polynomial basis*. Here we give a brief review to polynomial basis. Some of the other bases are discussed in the next chapter.

Polynomial Basis

Consider a finite field $GF(2^n)$ with an irreducible polynomial $f(x)$. If α be a root of this irreducible polynomial, then the set $\{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$ forms a polynomial basis. In polynomial basis representation, any element a of the field is represented as:

$$a = a_{n-1}\alpha^{n-1} + a_{n-2}\alpha^{n-2} + \dots + a_1\alpha + a_0 = \sum_{i=0}^{n-1} a_i\alpha^i \quad (2.9)$$

It should be noted that since we are not interested in evaluating a for a specific value of α , other variable symbols can be used instead of α too. As you have mentioned in the previous example we represented the elements of $GF(2^3)$ as $a_2x^2 + a_1x + a_0$, but by x we mean the root of the irreducible polynomial.

2.7 Finite Field Multiplication in Polynomial Basis

Consider a Finite field $GF(2^n)$ with the irreducible polynomial $f(x)$ as:

$$f(x) = x^n + \sum_{i=1}^{n-1} f_i x^i + 1 \quad (2.10)$$

where $f_i \in GF(2)$

Consider a and b as two elements of this field. By employing polynomial basis we have:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i \quad \text{and} \quad B(x) = \sum_{i=0}^{n-1} b_i x^i \quad (2.11)$$

The multiplication operation is then defined as:

$$C(x) = A(x)B(x) \bmod f(x) = \sum_{i=0}^{n-1} c_i x^i \quad (2.12)$$

This operation is performed in two steps:

1. Polynomial multiplication
2. Reduction modulo irreducible polynomial

Step 1:

In this step $A(x)$ is multiplied by $B(x)$ resulting in $D(x)$, considering the polynomial multiplication rules mentioned in section (2.4):

$$D(x) = A(x)B(x) = \sum_{i=0}^{2n-2} d_i x^i \quad (2.13)$$

The coefficients of $D(x)$ are calculated as follows:

$$d_i = \begin{cases} \sum_{j=0}^i a_j b_{i-j} & \text{for } 0 \leq i \leq n-1 \\ \sum_{j=i-n+1}^{n-1} a_j b_{i-j} & \text{for } n \leq i \leq 2n-2 \end{cases} \quad (2.14)$$

This polynomial operation can be represented in a matrix form as:

$$\mathbf{D}_{(2n-1) \times 1} = \mathbf{M}_{(2n-1) \times n} \mathbf{B} \quad (2.15)$$

Where \mathbf{D} is the coefficient column vector of $D(x)$, \mathbf{B} is the coefficient column vector of $B(x)$, and \mathbf{M} is the *Multiplication Matrix* which contains coefficients of $A(x)$ with the following form:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \\ d_{n+1} \\ \vdots \\ d_{2n-3} \\ d_{2n-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_0 & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 & a_0 \\ 0 & a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{n-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} & a_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1} \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} \quad (2.16)$$

Step 2

The result of the previous step was the polynomial $D(x)$ which obviously has a degree bigger than $n - 1$. Therefore it needs reduction. By reduction, we want to reduce the degree of $D(x)$ from $2n - 2$ to $n - 1$. This reduction process is defined as follows:

$$\begin{aligned} C(x) &= D(x) \bmod f(x) \\ &= \sum_{i=0}^{n-1} d_i x^i \bmod f(x) \\ &= \sum_{i=0}^{n-1} d_i x^i + \sum_{k=0}^{n-2} d_{k+n} x^{k+n} \bmod f(x) \end{aligned} \quad (2.17)$$

It is clear from the above equation that the reduction step is only performed on the elements of $D(x)$ which have a power greater than n .

Here a *Transfer Matrix*, $\mathbf{T}_{(n-1) \times n}$, is defined in a way that:

$$\begin{bmatrix} x^n \\ x^{n+1} \\ \vdots \\ x^{2n-3} \\ x^{2n-2} \end{bmatrix} = \mathbf{T} \times \begin{bmatrix} x^{n-1} \\ x^{n-2} \\ \vdots \\ x \\ 1 \end{bmatrix} \quad (2.18)$$

Clearly the role of this matrix is to find the residue of each x^{k+n} , for $0 \leq k \leq n-2$, modulo $f(x)$. The entries of this matrix are either “1” or “0”, and the form of this matrix directly depends on the irreducible polynomial. As an example, consider finite field $GF(2^5)$ with the irreducible polynomial $f(x) = x^5 + x^2 + 1$. Consider α as a root of $f(x)$. We have:

$$\alpha^5 + \alpha^2 + 1 = 0 \Rightarrow \alpha^5 = \alpha^2 + 1 \quad (2.19)$$

Note that since we are in finite field $GF(2^n)$, $-\alpha^5 = \alpha^5$. Multiplying both sides of (2.19) by α consequently, will result in:

$$\begin{cases} \alpha^6 = \alpha^3 + \alpha \\ \alpha^7 = \alpha^4 + \alpha^2 \\ \alpha^8 = \alpha^5 + \alpha^3 = \alpha^2 + 1 + \alpha^3 \end{cases} \quad (2.20)$$

And matrix \mathbf{T} is built as:

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

So by using (2.18) we can write:

$$x^{k+n} \bmod f(x) = \sum_{j=0}^{n-1} t_{k,j} x^{n-j-1} \quad (2.21)$$

And by substituting (2.21) in (2.17) we have:

$$\begin{aligned}
C(x) &= \sum_{i=0}^{n-1} d_i x^i + \sum_{k=0}^{n-2} d_{n+k} \sum_{j=0}^{n-1} t_{k,j} x^{n-j-1} \\
&= \sum_{i=0}^{n-1} d_i x^i + \sum_{j=0}^{n-1} \sum_{k=0}^{n-2} d_{n+k} t_{k,n-j-1} x^i
\end{aligned} \tag{2.22}$$

Hence, the final result is calculated.

2.8 Multiplier Architecture and Complexity

Hardware realization of (2.16) and (2.22) leads us to an actual circuit which is capable of performing finite field multiplication. This circuit will receive coefficients of two elements of the finite field, a and b , at the input and gives the coefficients of c at the output. Since our finite field is of the form $GF(2^n)$, the operations on the coefficients are performed in $GF(2)$. Therefore the multiplication operation is realized as a 2-input AND gate, while the addition operation is realized with a 2-input XOR gate.

In different hardware architectures for finite field multipliers, the important factor is to minimize the complexities. By *complexity* we mean two things:

- Space Complexity
- Time Complexity

Space complexity is the total number of gates used in the hardware realization of the multiplier. Since we just use two types of gates; i.e. 2-input AND gates, and 2-input XOR gates, space complexity is expressed in terms of the total number of these two gates.

Time complexity is the actual delay of the circuit. In other words, if the two inputs a , b be present at time t_0 , the output will be ready at time $t_0 + t_{\text{delay}}$. The source of the delay in the

circuit is the delay of the gates used in circuit. Again, since we just have two types of gates in the circuit, time complexity is expressed in terms of the delay of a 2-input AND gate, T_A , and the delay of a 2-input XOR gate, T_X .

When realizing the circuit for (2.16), due to the shape of matrix \mathbf{M} we need n^2 AND gates and $(n-1)^2$ XOR gates. All of the AND operations can be done in parallel so the total delay of the AND gates would be one T_A . The biggest XOR chain happens when realizing $(a_0 \ a_1 \ \dots \ a_{n-1}) \ (b_0 \ b_1 \ \dots \ b_{n-1})^T$. This operation needs $n-1$ XOR gates. By using a binary tree structure, the total delay of this operation would be $\lceil \log_2 n-1 \rceil$. Space and time complexity of realization of (2.22) directly depends on the form of the irreducible polynomial of the field and we will discuss it in the next chapter.

3 An Overview of Finite Field Multipliers

In this chapter we give a brief overview of the state of the art research in the field of finite field multiplication.

Finite field multipliers can be categorized based on their hardware architecture into three types:

- Parallel Multipliers
- Serial Multipliers
- Hybrid Multipliers

Bit parallel (or full parallel) finite field multipliers are the main subject of this thesis. In these multipliers, inputs and outputs are presented at the input ports in parallel at once. The output of the multiplier is ready at the output port after some delay. But no clock cycle is needed for performing multiplication operation.

On the other hand, bit serial multipliers receive the input bits in serial, and also perform the multiplication operation serially. So the output will be ready after some clock cycles, bit by bit. Hybrid multipliers can receive the inputs serially, but they may perform the multiplication operation in parallel for a word size of inputs.

In this chapter we first thoroughly review the best bit parallel finite field multipliers. Then we will briefly review some serial multiplier architectures.

3.1 Parallel Finite Field Multipliers

One of the most important contributions to this field of research is the work of Mastrovito in [21]. In the previous chapter we showed the actual multiplication of two elements in a finite field of form $GF(2^n)$ regarding the polynomial basis. That method is called *conventional* multiplication and has two steps: the first step is multiplying two polynomials and the second step is reducing the result of the polynomial multiplication by the irreducible polynomial. On the other hand, Mastrovito has introduced another method in [21] which has been called *mastrovito multiplier* in literature. Mastrovito multiplier combines those two steps in just one matrix multiplication. The whole concept of mastrovito multiplier is to find a matrix \mathbf{M} (later called matrix \mathbf{Z}) which satisfies the following equation:

$$\mathbf{C} = \mathbf{M}\mathbf{B} \quad (3.1)$$

where

$$\mathbf{M} = \begin{bmatrix} f_{0,0} & \cdots & f_{0,n-1} \\ \vdots & \ddots & \vdots \\ f_{n-1,0} & \cdots & f_{n-1,n-1} \end{bmatrix} \quad (3.2)$$

The entries of matrix \mathbf{M} depend on the coefficients of $A(x)$ and on the coefficients of the \mathbf{T} matrix as follows:

$$f_{i,j} = \begin{cases} a_i & \text{for } j = 0 \text{ and } 0 \leq i \leq n-1 \\ u(i-j)a_{i-j} + \sum_{k=0}^{j-1} t_{j-1-k,i}a_{n-1-k} & \text{for } 1 \leq j \leq n-1 \text{ and } 1 \leq i \leq n-1 \end{cases} \quad (3.3)$$

In the above formula u is a step function defined as:

$$u(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases} \quad (3.4)$$

It is important to note that although mastrovito multiplier is referred to as a one step multiplication method considering the conventional two step method, but it also has two steps itself. The first step is to build up the \mathbf{M} matrix, and the second step is to perform the matrix multiplication. Mastrovito multiplier has been used as the base of many efficient finite field multiplication methods since it was presented.

Another important contribution in this field was the work of Paar in [27]. Paar combined the conventional multiplier with *Karatsuba-Ofman* algorithm (KOA) in order to gain more reductions in space complexity. KOA was first introduced in [14] as a method for multiplying large integers. KOA is a *divide and conquer* form algorithm that divides the operands into two parts with less number of digits (half number of digits) and forms the final result with the help of the product of these parts. A good explanation of KOA algorithm can be found in [15]. This idea can directly be applied on polynomial basis representation of finite fields since each element can be represented by the coefficients which are in the field of $GF(2)$; i.e. $\{0,1\}$. Therefore the representation of the field's elements would be a string of bits.

Regarding the conventional method of multiplication, KOA is a method for improving the first step. Applying KOA to polynomial multiplication would be recursive if the degree of the polynomials is $n - 1$ and n is a power of 2. Let's consider two elements a and b in a finite field $GF(2^n)$:

$$\begin{aligned} A(x) &= a_{n-1}x^{n-1} + \dots + a_1x + a_0 \\ B(x) &= b_{n-1}x^{n-1} + \dots + b_1x + b_0 \end{aligned}$$

Immediately we can rephrase $A(x)$ and $B(x)$ as below:

$$\begin{aligned}
A(x) &= x^{\frac{n}{2}} \left(a_{n-1} x^{\frac{n}{2}-1} + \dots + a_{(\frac{n}{2})} \right) + \left(a_{(\frac{n}{2}-1)} x^{\frac{n}{2}-1} + \dots + a_1 x + a_0 \right) = x^{\frac{n}{2}} A_H + A_L \\
B(x) &= x^{\frac{n}{2}} \left(b_{n-1} x^{\frac{n}{2}-1} + \dots + b_{(\frac{n}{2})} \right) + \left(b_{(\frac{n}{2}-1)} x^{\frac{n}{2}-1} + \dots + b_1 x + b_0 \right) = x^{\frac{n}{2}} B_H + B_L
\end{aligned} \tag{3.5}$$

Therefore with this new notation, we can change the method of multiplying those two polynomials, from the conventional multiplication method, which involves multiplying each element in $A(x)$ to all the elements in $B(x)$, to a new method as below:

$$\begin{aligned}
A(x)B(x) &= \left(x^{\frac{n}{2}} A_H + A_L \right) \left(x^{\frac{n}{2}} B_H + B_L \right) \\
&= x^n A_H B_H + x^{\frac{n}{2}} \{ (A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L) \} + A_L B_L
\end{aligned} \tag{3.6}$$

It is obvious that subtraction in the second term is the same as addition since we are using the fields of characteristic 2. The gate complexities for conventional method as mentioned in chapter 2 are:

$$\begin{cases} \#AND = n^2 \\ \#XOR = (n-1)^2 \end{cases} \tag{3.7}$$

But the corresponding complexities for the new method are:

$$\begin{cases} \#AND = \frac{3}{4} n^2 \\ \#XOR = \frac{3}{4} n^2 + n - 1 \end{cases} \tag{3.8}$$

Further reductions can be gained by applying the same method on the polynomial multiplications: $A_H B_H$, $A_L B_L$, and $(A_H + A_L)(B_H + B_L)$. It is obvious that this procedure of splitting polynomials can be applied recursively in i steps, where $i = \log_2 n$. It has been mentioned in [27] that the total space complexity would be:

$$\begin{cases} \#AND = n^{\log_2 3} \\ \#XOR = 6n^{\log_2 3} - 8n + 2 \end{cases} \tag{3.9}$$

The method of Parr in [27] was widely used later for building sub-quadratic space complexity multipliers.

Fenn et al presented a parallel *dual basis* finite field multiplier in [8]. First we have to define dual basis: A set of n elements $\{\beta_0, \beta_1, \dots, \beta_{n-1}\}$ forms a basis for $GF(2^n)$ if the β_i s are linearly independent over the field $GF(2)$. Let $f(x)$ be an irreducible polynomial of degree n , then as we mentioned in chapter 2, the polynomial basis of the field would be $\{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$, considering α as a root of $f(x)$. The trace of $\beta \in GF(2^n)$ relative to the subfield $GF(2)$ is defined as:

$$Tr(\beta) = \sum_{i=0}^{n-1} \beta^{2^i} \quad (3.10)$$

This trace function is a *linear mapping* from the finite field $GF(2^n)$ onto the finite field $GF(2)$. Let $\{\alpha_0, \alpha_1, \dots, \alpha_{n-1}\}$ and $\{\beta_0, \beta_1, \dots, \beta_{n-1}\}$ be any two bases for $GF(2^n)$ and also let $\gamma \in GF(2^n)$ with $\gamma \neq 0$. Then these two bases are said to be dual with respect to γ if:

$$Tr(\gamma \alpha_i \beta_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (3.11)$$

If γ is a fixed non-zero element of the field, and if $\{\beta_0, \beta_1, \dots, \beta_{n-1}\}$ is a dual basis of the polynomial basis, $\{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$, then any element a of the field can be expressed as polynomial basis or dual basis as follows:

$$a = \sum_{i=0}^{n-1} a_i \alpha^i = \sum_{i=0}^{n-1} a_i^* \beta_i \quad (3.12)$$

In order to find a_j^* s we use (3.11):

$$Tr(\gamma \alpha^j a) = Tr\left(\gamma \alpha^j \sum_{i=0}^{n-1} a_i^* \beta_i\right) = \sum_{i=0}^{n-1} a_i^* Tr(\gamma \alpha^j \beta_i) = a_j^* \quad (3.13)$$

Now consider two elements a, b in the field. We want to find $c = a \times b \bmod f(x)$. Here a is represented over dual basis and b is represented over polynomial basis as:

$$\begin{cases} a = \sum_{i=0}^{n-1} a_i^* \beta_i \\ b = \sum_{i=0}^{n-1} b_i \alpha^i \end{cases} \quad (3.14)$$

Then c can be obtained as follows:

$$\begin{bmatrix} c_0 \\ c \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} a_0^* & a_1^* & \cdots & a_{n-1}^* \\ a_1^* & a_2^* & \cdots & a_n^* \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1}^* & a_n^* & \cdots & a_{2n-2}^* \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} \quad (3.15)$$

In the above formula a_i^* s for $n \leq i \leq 2n-2$ are recursively calculated as follows:

$$a_{n+k}^* = \sum_{j=0}^{n-1} f_j a_{j+k}^* \quad \text{for } 0 \leq k \leq n-2 \quad (3.16)$$

In which, f_j s are the coefficients of the irreducible polynomial of the field. From (3.15) we have:

$$\begin{cases} c_0 = a_0^* b_0 + a_1^* b_1 + \cdots + a_{n-1}^* b_{n-1} \\ c_1 = a_1^* b_0 + a_2^* b_1 + \cdots + a_n^* b_{n-1} \\ \vdots \\ c_{n-1} = a_{n-1}^* b_0 + a_n^* b_1 + \cdots + a_{2n-2}^* b_{n-1} \end{cases} \quad (3.17)$$

From the above equations it can be seen that the n product bits are generated by n identical functions of the form:

$$h(a, b) = a_k^* b_0 + a_{k+1}^* b_1 + \cdots + a_{k+n-1}^* b_{n-1} \quad \text{for } 0 \leq k \leq n-1 \quad (3.18)$$

Therefore a bit-parallel dual basis multiplier for $GF(2^n)$ can be constructed out of n $GF(2)$ inner product modules that implement (3.18), and one other module that generates the a_{n+k}^* for $0 \leq k \leq n-2$ from (3.16).

Koc and Sunar presented two new finite field multipliers for all one polynomials (AOP) in [17]. The first multiplier was a polynomial basis multiplier based on the mastrovito multiplier with a slight change, and the second one was a *normal basis* multiplier.

An AOP over the finite field of $GF(2^n)$ is a degree- n polynomial with all the coefficients as 1; i.e. $f(x) = x^n + x^{n-1} + \dots + x + 1$. According to (3.2) and (3.3) the mastrovito matrix of an AOP of degree n would be as follows:

$$\mathbf{Z} = \begin{bmatrix} a_0 & a_{n-1} & a_{n-1} + a_{n-2} & a_{n-2} + a_{n-3} & \cdots & a_2 + a_1 \\ a_1 & a_0 + a_{n-1} & a_{n-2} & a_{n-1} + a_{n-3} & \cdots & a_3 + a_1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-2} & a_{n-3} + a_{n-1} & a_{n-4} + a_{n-2} & a_{n-5} + a_{n-3} & \cdots & a_1 \\ a_{n-1} & a_{n-2} + a_{n-1} & a_{n-3} + a_{n-2} & a_{n-4} + a_{n-3} & \cdots & a_0 + a_1 \end{bmatrix} \quad (3.19)$$

This matrix can be decomposed into matrices \mathbf{Z}_1 and \mathbf{Z}_2 as $\mathbf{Z} = \mathbf{Z}_1 + \mathbf{Z}_2$:

$$\mathbf{Z}_1 = \begin{bmatrix} a_0 & 0 & a_{n-1} & a_{n-2} & \cdots & a_2 \\ a_1 & a_0 & 0 & a_{n-1} & \cdots & a_3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & a_{n-5} & \cdots & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_0 \end{bmatrix} \quad \mathbf{Z}_2 = \begin{bmatrix} 0 & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 \\ 0 & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 \\ 0 & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 \end{bmatrix} \quad (3.20)$$

Now in order to compute $\mathbf{C} = \mathbf{ZB} = (\mathbf{Z}_1 + \mathbf{Z}_2)\mathbf{B}$, first we compute $\mathbf{D} = \mathbf{Z}_1\mathbf{B}$ and $\mathbf{E} = \mathbf{Z}_2\mathbf{B}$ in parallel, and then we compute the result $\mathbf{C} = \mathbf{D} + \mathbf{E}$. By using this method, time delay of the circuit would be less than a regular mastrovito multiplier for AOP.

Before reviewing the second multiplier presented in [17] we have to explain normal basis. A set N of the form $N = \{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{n-1}}\}$ in a finite field $GF(2^n)$ forms a normal basis, where β is the root of the irreducible polynomial. Since the irreducible polynomial of the field is an AOP we have:

$$\beta^{n+1} = 1 \quad (3.21)$$

Further more if 2 is primitive in \mathbf{Z}_{n+1} then we have:

$$N = \{\beta, \beta^2, \beta^3, \dots, \beta^n\} \quad (3.22)$$

Set N in (3.22) is also a basis. It is actually a shifted version of the polynomial basis. Any element a of $GF(2^n)$ represented with normal basis can easily be converted to this shifted polynomial basis (SPB) representation as follows:

$$a = \sum_{i=0}^{n-1} a_i \beta^{2^i} = \sum_{i=0}^{n-1} a'_i \beta^i \quad (3.23)$$

The coefficients of the shifted polynomial basis can be obtained as:

$$a'_{2^i \bmod (n+1)} = a_i \quad \text{for} \quad 0 \leq i \leq n-1 \quad (3.24)$$

In this multiplier, the two inputs a and b are first converted from normal basis to shifted polynomial basis, then a regular polynomial basis multiplication is performed. At the end of this computation the result is obtained as $g = ab/\beta^2$:

$$\begin{aligned} g &= g_{n-1}\beta^{n-1} + \dots + g_1\beta + g_0 \\ &= (d_{n-1} + e)\beta^{n-1} + \dots + (d_1 + e)\beta + (d_0 + e) \end{aligned} \quad (3.25)$$

Note that $d_k = \mathbf{Z}_1(k, :) \mathbf{B}$ for $0 \leq k \leq n-1$ and $e = \mathbf{Z}_2(0, :) \mathbf{B} = \mathbf{Z}_2(1, :) \mathbf{B} = \dots = \mathbf{Z}_2(n-1, :) \mathbf{B}$, and $\mathbf{X}(j, :)$ is the j 'th row of matrix \mathbf{X} .

Then we have to multiply g by β^2 :

$$h = (d_{n-1} + e)\beta^{n+1} + (d_{n-2} + e)\beta^n + \dots + (d_1 + e)\beta^3 + (d_0 + e)\beta^2 \quad (3.26)$$

Since $\beta^{n+1} = \beta + \beta^2 + \dots + \beta^n$, the coefficient $(d_{n-1} + e)$ is added to all the other coefficients. Therefore we can write h in shifted polynomial basis as:

$$h = (d_{n-2} + d_{n-1})\beta^n + \dots + (d_1 + d_{n-1})\beta^3 + (d_0 + d_{n-1})\beta^2 + (d_{n-1} + e)\beta \quad (3.27)$$

In order to convert h in SPB to the final result, c , in normal basis we just apply the reverse of (3.23).

When mastrovito multiplier was introduced in [21], it was only analyzed for trinomials of type $x^n + x + 1$. The space complexity of this multiplier was: n^2 AND gates, and $n^2 - 1$ XOR gates. Sunar and Koc presented a new formulation of the multiplication matrix in [37]. With the help of this formulation, new multiplier architecture was presented for mastrovito multiplier, in which the XOR complexity was $n^2 - 1$ for all trinomials of type $x^n + x^t + 1$ for $0 \leq t < n$ and $n^2 \neq 2t$. Furthermore it was proved that if $n = 2t$, then the XOR complexity would be $n^2 - n/2$.

As we explained in chapter 2, in a finite field $GF(2^n)$ with $f(x)$ as the irreducible polynomial in polynomial basis, in order to find $c = ab \bmod f(x)$ we first find $d = ab$; $D(x)$ is a polynomial of degree $2n - 2$. In the next step $D(x)$ is reduced by the $f(x)$ to $C(x)$ which is a polynomial of degree $n - 1$. Matrix representation of the first step as described in (2.16) would be:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{n-2} \\ d_{n-1} \\ d_n \\ d_{n+1} \\ \vdots \\ d_{2n-3} \\ d_{2n-2} \end{bmatrix} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_0 & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 & a_0 \\ 0 & a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{n-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} & a_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix}$$

It is important to note that, not all the elements of d need reduction in the second step, but only the elements of d that have a power of x bigger than $n - 1$. The number of reductions for a specific element depends on the degree of the element and on the value of the middle term of the irreducible trinomial; i.e. t . The maximum number of reductions is performed on the highest order element d_{2n-2} . Let k be the number of reductions required for this element. This integer k has the property $2n - 2 - k(n - t) < n$, which implies $k > \frac{n-2}{n-t}$. Therefore, we have:

$$k = \left\lceil \frac{n-2}{n-t} \right\rceil + 1 \quad (3.28)$$

The objective of [37] is to obtain the matrix $Z_n \times n$ by systematically reducing the last $n - 1$ rows of the matrix $M_{(2n-1) \times n}$ using a general trinomial. In order to accomplish this task, a reduction array was defined as below:

$$\begin{array}{rcll}
x^n & = & 1 & + x^t \\
x^{n+1} & = & x & + x^{t+1} \\
\vdots & & & \\
x^{2n-t-1} & = & x^{n-t-1} & + x^{n-1} \\
x^{2n-t} & = & x^{n-t} & + 1 + x^t \\
\vdots & & & \\
x^{3n-2t-1} & = & x^{2n-2t-1} & + x^{n-t-1} + x^{n-1} \\
x^{3n-2t} & = & x^{2n-2t} & + x^{n-t} + 1 + x^t \\
\vdots & & & \\
x^{kn-(k-1)t} & = & x^{(k-1)n-(k-1)t} & + x^{(k-2)n-(k-2)t} + \dots + 1 + x^t \\
\vdots & & & \\
x^{2n-2} & = & x^{n-2} & + x^{t-2} + \dots + x^{(k-1)n-(k-2)t-2} + x^{kn-(k-2)t-2}
\end{array}$$

Figure 3.1 Reduction Array for a General Trinomial

The columns of the reduction array have the following properties:

- The first column on the right-hand side is the sequence $1, x, x^2, \dots, x^{n-t}$;
- The second column contains two sequences: The sequence $x^t, x^{t+1}, \dots, x^{n-t}$ followed by the sequence $1, \dots, x^{n-t-1}, x^{n-t}, \dots, x^{n-2}$;
- For $3 \leq i \leq n-t$, the i 'th column is obtained by shifting down the $(i-1)$ 'th column $m-t$ positions;

It is also seen that the reduction array can be divided into k partitions, and each partition has $n-t$ terms except for the last partition which may have less terms. If the partitions are enumerated in increasing order, beginning from the topmost as the 0th partition, the i 'th partition will consist of the rows starting with the term $x^{n+i(n-t)}$ and ending with the term $x^{n+(i+1)(n-t)-1}$.

In the reduction process the rows defined by the reduction array are added to the rows of matrix \mathbf{M} in order to eliminate the last $n-t$ rows of \mathbf{M} . The exponent on the left-hand side provides the index to the source row, and the exponents on the right-hand side provide the destination. For example row n is added to row 0 and row t .

In order to construct the \mathbf{Z} matrix the following formulation is used:

$$\mathbf{Z} = \mathbf{X} + \mathbf{Y} \quad (3.29)$$

Where matrix \mathbf{X} is the upper n rows of matrix \mathbf{M} (destination rows) and matrix \mathbf{Y} is the contribution of the $n - 1$ lower rows of matrix \mathbf{M} (source rows):

$$\mathbf{X} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_0 & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 & a_0 \end{bmatrix} \quad (3.30)$$

Since we have two types of sequences on the right hand side columns: those of form $1, x, x^2, \dots, x^{n-2}$ or $1, \dots, x^{n-t-1}, x^{n-t}, \dots, x^{t-2}$, and those of form $x^t, x^{t+1}, \dots, x^{n-1}$, we can write:

$$\mathbf{Y} = \mathbf{T} + \mathbf{U} \quad (3.31)$$

Where \mathbf{T} is the contribution of the first type of sequences, and \mathbf{U} is the contribution of the second type of sequences. The first column is the sequence $1, x, x^2, \dots, x^{n-2}$. So we have to add the rows $\mathbf{M}(n, :)$, $\mathbf{M}(n+1, :)$, \dots , $\mathbf{M}(2n-2, :)$ to $\mathbf{Z}(0, :)$, $\mathbf{Z}(1, :)$, \dots , $\mathbf{Z}(n-2, :)$. This contribution of the first column is shown in the matrix below:

$$\mathbf{T}_0 = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \cdots & 0 & a_{m-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \quad (3.32)$$

The second sequence in the second column is $1, \dots, x^{n-t-1}, x^{n-t}, \dots, x^{t-2}$. So we add the rows $\mathbf{M}(2n-t, :), \mathbf{M}(2n-t+1, :), \dots, \mathbf{M}(2n-2, :)$ to $\mathbf{Z}(0, :), \mathbf{Z}(1, :), \dots, \mathbf{Z}(t-2, :)$. This can be shown by shifting up matrix \mathbf{T}_0 $n-t$ rows: $\mathbf{T}_1 = \mathbf{T}_0[\uparrow(n-t)]$. By similarity we have:

$$\begin{aligned} \mathbf{T}_i &= \mathbf{T}_{i-1}[\uparrow(n-t)] = \mathbf{T}_0[\uparrow i(n-t)] \\ \mathbf{T} &= \mathbf{T}_0 + \mathbf{T}_1 + \dots + \mathbf{T}_{k-1} = \sum_{i=0}^{k-1} \mathbf{T}_0[\uparrow i(n-t)] \end{aligned} \quad (3.33)$$

The first sequence in the second column is the sequence $x^t, x^{t+1}, \dots, x^{n-1}$. So we have to add the rows $\mathbf{M}(n, :), \mathbf{M}(n+1, :), \dots, \mathbf{M}(2n-2, :)$ to the rows $\mathbf{Z}(t, :), \mathbf{Z}(t+1, :), \dots, \mathbf{Z}(n-1, :)$. This is done with the help of matrix \mathbf{U}_0 :

$$\mathbf{U}_0 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & a_{n-1} & a_{n-2} & \cdots & a_t & \cdots & a_2 & a_1 \\ 0 & 0 & a_{n-1} & \cdots & a_{t+1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} & \cdots & a_{n-t+1} & a_{n-t} \end{bmatrix} \begin{matrix} 0 \\ \vdots \\ t-1 \\ t \\ t+1 \\ \vdots \\ n-1 \end{matrix} \quad (3.34)$$

The first sequence in the third column is $x^t, x^{t+1}, \dots, x^{n-1}$. So we add the rows $\mathbf{M}(2n-t, :), \mathbf{M}(2n-t+1, :), \dots, \mathbf{M}(2n-2, :)$ to the rows $\mathbf{Z}(t, :), \mathbf{Z}(t+1, :), \dots, \mathbf{Z}(n-1, :)$. This can be shown by shifting right matrix \mathbf{U}_0 $n-t$ columns: $\mathbf{U}_1 = \mathbf{U}_0[\rightarrow(n-t)]$. By similarity we have:

$$\begin{aligned} \mathbf{U}_i &= \mathbf{U}_{i-1}[\rightarrow(n-t)] = \mathbf{U}_0[\rightarrow i(n-t)] \\ \mathbf{U} &= \mathbf{U}_0 + \mathbf{U}_1 + \dots + \mathbf{U}_{k-1} = \sum_{i=0}^{k-1} \mathbf{U}_0[\rightarrow i(n-t)] \end{aligned} \quad (3.35)$$

So with the help of these matrices, the mastrovito matrix is built as:

$$\begin{aligned}
\mathbf{Z} &= \mathbf{X} + \mathbf{Y} \\
&= \mathbf{X} + \mathbf{T} + \mathbf{U} \\
&= \mathbf{X} + \sum_{i=0}^{k-1} \mathbf{T}_0 [\uparrow i(n-t)] + \sum_{i=0}^{k-1} \mathbf{U}_0 [\rightarrow i(n-t)]
\end{aligned} \tag{3.36}$$

Mastrovito multiplier did not show promising results for finite fields defined by high hamming weight (the number of nonzero coefficients) irreducible polynomials. As a solution, Song and Parhi presented a modified mastrovito multiplier in [36]. With this architecture, it is possible to build finite field multipliers for $GF(2^n)$ with XOR complexity proportional to $n - l - pwt$, in which pwt is the hamming weight of the irreducible polynomial.

In order to compute $c = ab$ in finite field $GF(2^n)$ with modified mastrovito multiplier, first a modified mastrovito matrix \mathbf{U} is built and $\mathbf{D} = \mathbf{U}\mathbf{B}$ is computed as follows:

$$\mathbf{U}_{(n+1) \times n} \mathbf{B} = \begin{bmatrix} u_{n-1}^0 & u_{n-1}^1 & \cdots & u_{n-1}^{n-1} \\ u_{n-2}^0 & u_{n-2}^1 & \cdots & u_{n-2}^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ u_0^0 & u_0^1 & \cdots & u_0^{n-1} \\ u_{-1}^0 & u_{-1}^1 & \cdots & u_{-1}^{n-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} d_{n-1} \\ d_{n-2} \\ \vdots \\ d_0 \\ d_c \end{bmatrix} = \mathbf{D} \tag{3.37}$$

The coefficients of matrix \mathbf{U} are calculated as follows:

$$\begin{cases} u_i^0 = a_i & for \quad 0 \leq i \leq n-1 \\ u_i^k = u_{i-1}^{k-1} + (1 + f_i)(u_{n-1}^{k-1} - u_{n-1}^{k-2}) & for \quad 0 \leq i, k \leq n-1 \\ u_{-1}^0 = 0 \\ u_{-1}^k = u_{n-1}^{k-1} & for \quad 0 \leq k \leq n-1 \end{cases} \tag{3.38}$$

where f_i s are the coefficients of the irreducible polynomial. Then $c = ab$ can be obtained as follows:

$$\begin{bmatrix} c_{n-1} \\ c_{n-2} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} d_{n-1} + d_c \\ d_{n-2} + d_c \\ \vdots \\ d_0 + d_c \end{bmatrix} \quad (3.39)$$

Detailed proof of (3.39) can be found in [36].

Following the work of Sunar and Koc in [37], Halbutogullari and Koc tried to find a general formulation for mastrovito multiplier with any type of irreducible polynomial in [9]. Although this formulation covers all the polynomials of the form:

$$f(x) = x^{n_k} + x^{n_{k-1}} + \dots + x^{n_1} + x^{n_0}$$

where $n = n_k > n_{k-1} > \dots > n_1 > n_0 = 0$, but it does not offer a practical way of hardware implementation of the multiplier. However it analyzes the mastrovito multiplier for an equally spaced polynomial (ESP) and shows very promising complexities for this type of irreducible polynomials. In a finite field $GF(2^n)$ an irreducible ESP has the following form:

$$f(x) = x^{k\Delta} + x^{(k-1)\Delta} + \dots + 1 \quad (3.40)$$

where $k\Delta = n$. the space complexity of the mastrovito multiplier for such polynomial would be:

$$\begin{cases} \#AND = n^2 \\ \#XOR = n^2 - \Delta \end{cases} \quad (3.41)$$

A special case of ESP is the trinomial of the form $x^n + x^{\frac{n}{2}} + 1$. The complexity of XOR gates for a multiplier with this polynomial would be $\#XOR = n^2 - n/2$. To the best of our knowledge, so far this is the lowest complexity of a finite field multiplier in literature. It has been proved in [9] that a mastrovito multiplier cannot have space complexity better than this:

$$\begin{cases} \#AND = n^2 \\ \#XOR = n^2 - \frac{n}{2} \end{cases} \quad (3.42)$$

Although [9] provided a general formulation for mastrovito multiplier, a systematic method of designing this multiplier was not available until Zhang and Parhi presented an explicit algorithm for efficiently designing that in [41].

Considering finite field $GF(2^n)$ with a general irreducible polynomial $f(x)$ we have:

$$\begin{cases} f(x) = x^n + f_1 x^{n-1} + \dots + f_1 x + f_0 \\ f_i \in \{0,1\} \quad \text{for} \quad 0 \leq i \leq n-1 \\ f_0 = 1 \end{cases} \quad (3.43)$$

(3.43) can also be written as:

$$\begin{cases} f(x) = x^n + x^{k_s} + x^{k_{s-1}} + \dots + x^{k_1} + 1 \\ n > k_s > k_{s-1} > \dots > k_1 > 1 \end{cases} \quad (3.44)$$

Now a set N can be built with the following algorithm from [41]:

Input: The parameters of irreducible polynomial: n, k_1, \dots, k_s

Output: set $N \subset \{0,1,\dots,n-2\}$

Procedure:

Step 1: Generate a weighted tree D according to the following properties:

- *Each node d_j in D has at most s child nodes and each edge has the weight $w \in \{(n - k_i) \mid 1 \leq i \leq s\}$*
- *Let d_1 denote the root and $h(d_1, d_j)$ denote the weight of path from d_1 to d_j , where $h(d_1, d_1) = 0$; for all d_j , if $\exists r \in \{(n - k_i) \mid 1 \leq i \leq s\}$ and $(h(d_1, d_j) + r) < n - 1$, then d_j has a child node d_l with an edge of weight r*
- *For all d_j , $h(d_1, d_j) < n - 1$*

Step 2: Construct a multi-set $H = \{h(d_1, d_j), \forall d_j \in D\}$ and set $N = \emptyset$

Step 3: For $0 \leq j \leq n-2$, do

- a. create multi-set $S_j = \emptyset$*
- b. $\forall h \in H$, if $h = j$, then insert h into S_j*
- c. If $(|S_j| \bmod 2) = 1$, then insert j into N*

In the above algorithm, a multi-set is like a set, except that repeated elements are allowed, and $|S_j|$ represents the order of S_j . From the above algorithm, we know that the least two elements in N are always 0 and $(n - k_s)$ and we have $|N| \leq k_s$.

The total space complexity of the mastrovito multiplier proposed in [41] would be:

$$\begin{cases} \#AND = n^2 \\ \#XOR = (n + s - 1)(n - 1) + \sum_{j \in N} (n - j - 1) \end{cases} \quad (3.45)$$

It was known by the previous works that the best complexities for finite field multipliers will be obtained when the irreducible polynomial of the field is either trinomial, AOP, or ESP. However it is not possible to find such irreducible polynomial for any value of n . So the next best choice was a pentanomial. Rodriguez and Koc presented a mastrovito multiplier for a special type of pentanomials in [34]. This special type of pentanomial has the following form:

$$f(x) = x^n + x^{m+1} + x^m + x + 1 \quad (3.46)$$

where $2 \leq m \leq \left\lfloor \frac{n}{2} \right\rfloor - 1$. In order to build the multiplier, the work presented in [37] is used. So first a reduction array should be generated:

$$\begin{aligned}
x^n &= 1 + x + x^m + x^{m+1} \\
x^{n+1} &= x + x^2 + x^{m+1} + x^{m+2} \\
x^{n+2} &= x^2 + x^3 + x^{m+2} + x^{m+3} \\
&\vdots \\
x^{2n-m-2} &= x^{n-m-2} + x^{n-m-1} + x^{n-2} + x^{n-1} \\
x^{2n-m-1} &= x^{n-m-1} + x^{n-m} + x^{n-1} + 1 + x + x^m + x^{m+1} \\
x^{2n-m} &= x^{n-m} + x^{n-m+1} + 1 + x^m + x^2 + x^{m+2} \\
x^{2n-m+1} &= x^{n-m+1} + x^{n-m+2} + x + x^{m+1} + x^3 + x^{m+3} \\
&\vdots \\
x^{2n-3} &= x^{n-3} + x^{n-2} + x^{m-3} + x^{2m-3} + x^{m-1} + x^{2m-1} \\
x^{2n-2} &= x^{n-2} + x^{n-1} + x^{m-2} + x^{2m-2} + x^m + x^{2m}
\end{aligned}$$

Figure 3.2 Reduction Array for $x^n + x^{m+1} + x^m + x + 1$

We can summarize the above equations based on the number of operand as:

$$x^{n+i} = \begin{cases} x^i + x^{i+1} + x^{m+i} + x^{m+i+1} & \text{for } 0 \leq i \leq n-m-2 \\ x^i + x^{i+1} + x^{m+i} + 1 + x + x^m + x^{m+1} & \text{for } i = n-m-1 \\ x^i + x^{i+1} + x^{i-(n-m)} + x^{i-n+2m} + x^{i-(n-m)+2} + x^{i-n+2m+2} & \text{for } n-m \leq i \leq n-2 \end{cases} \quad (3.47)$$

Now if we consider $D(x) = A(x)B(x)$, and $C(x) = D(x) \bmod f(x)$ then in order to obtain coefficients of $C(x)$ we just need to add the nonzero elements of each one of the n columns. For example, in order to obtain the first coordinate c_0 , we just need to add the nonzero coefficients of the first column to the first coordinate of the product polynomial d_0 :

$$c_0 = d_0 + d_n + d_{2n-m-1} + d_{2n-m} \quad (3.48)$$

The entire process will have total complexity as:

$$\begin{cases} \#AND = n^2 \\ \#XOR = n^2 + n \end{cases} \quad (3.49)$$

Chang et al presented a new way of representing finite field elements in [2] called *redundant representation*. Combining this representation with KOA helps achieving lower complexity multipliers for all one polynomials. Redundant representation of the elements of finite field $GF(2^n)$ was first used for building finite field multipliers in [13]. Considering an AOP in finite field $GF(2^n)$ as:

$$f(x) = \sum_{i=0}^n x^i$$

If α be a root of $f(x)$, then $\alpha^{n+1} = 1$ and $\alpha^n + \alpha^{n-1} + \dots + \alpha + 1 = 0$. A polynomial basis would be $\{1, \alpha, \dots, \alpha^{n-1}\}$. Now a redundant representation is obtained by expanding the polynomial basis as $\{1, \alpha, \dots, \alpha^{n-1}, \alpha^n\}$. Any element a in $GF(2^n)$ is represented as:

$$a = \sum_{i=0}^n a_i \alpha^i \quad (3.50)$$

Note that the redundant representation is not unique. For example $a_0 + a_1\alpha + \dots + a_n\alpha^n$ and $b_0 + b_1\alpha + \dots + b_n\alpha^n$ denote the same element if $b_i = a_i + a_m$. Considering the modular reduction in redundant representation, since $\alpha^{n+1} = 1$, for any element a of $GF(2^n)$ we have:

$$\alpha^i a = a_0 \alpha^i + a_1 \alpha^{i+1} + \dots + a_{n-i} \alpha^n + a_{n-i+1} + a_{n-i+2} \alpha + \dots + a_n \alpha^{i-1} \quad (3.51)$$

This means that $\alpha^i a$ can be computed by an i -fold *right cyclic shift* of a . Now consider $n = 2m$, any two elements a, b in $GF(2^n)$ can be partitioned in two parts as:

$$\begin{aligned} a &= \sum_{i=0}^{2m} a_i \alpha^i = \sum_{i=0}^{m-1} a_i \alpha^i + \alpha^m \sum_{i=0}^m a_{i+m} \alpha^i \\ &= A + \alpha^m B \end{aligned} \quad (3.52)$$

and

$$\begin{aligned}
b &= \sum_{i=0}^{2m} b_i \alpha^i = \sum_{i=0}^m b_i \alpha^i + \alpha^{m+1} \sum_{i=0}^{m-1} b_{i+m} \alpha^i \\
&= C + \alpha^{m+1} D
\end{aligned} \tag{3.53}$$

Then in order to obtain $c = ab$ we have:

$$\begin{aligned}
c &= ab = (A + \alpha^m B)(C + \alpha^{m+1} D) \\
&= AC + (BC + \alpha AD)\alpha^n + BD\alpha^{2m+1} \\
&= AC + BD\alpha^{2m+1} + ((A + B)(C + \alpha D) + AC + \alpha BD)\alpha^m \\
&= AC + BD\alpha^{m+1} + (AC + BD\alpha^{m+1})\alpha^m + (A + B)(C + \alpha D)\alpha^m
\end{aligned} \tag{3.54}$$

Computation of (3.54) needs the following number of gates:

$$\begin{cases} \#AND = 3m^2 + 4m + 1 = \frac{3}{4}n^2 + 2n + 1 \\ \#XOR = 3m^2 + 6m + 1 = \frac{3}{4}n^2 + 3n + 1 \end{cases} \tag{3.55}$$

So it can be seen that the number of all gates is reduced by 25%.

We have seen that finite field multipliers based on trinomials result in the best space complexity among different multipliers, and it was proved in [9] that no other multiplier can be built with better space complexity. But about the speed of the multiplier and time complexity, still some works have been done. Fan and Dai presented a new representation for finite field elements in [6], called *shifted polynomial basis* (SPB) which helps building faster finite field multipliers for trinomials. We have previously seen SPB in [8] when building finite field multiplier for fields based on AOP. But that was just a special case of shifted polynomial basis.

Let v be an integer and the set $M = \{x^i \mid 0 \leq i \leq n-1\}$ be a polynomial basis of $GF(2^n)$. Then the ordered set $x^{-v}M = \{x^{i-v} \mid 0 \leq i \leq n-1\}$ is called the shifted polynomial basis (SPB) with respect to M . let $f(x) = x^n + x^k + 1$ be the general trinomial defining this field. It is proved

in [6] that the best value for v is k . an element a of the field is represented in SPB as follows:

$$A(x) = x^{-v} \sum_{i=0}^{n-1} a_i x^i = \sum_{i=-v}^{n-1-v} a_{i+v} x^i \quad (3.56)$$

Let's review the basis conversion between PB and SPB first. Consider d and a be two elements of the field represented by PB and SPB respectively:

$$\begin{aligned} D(x) &= \sum_{i=0}^{n-1} d_i x^i = \sum_{i=0}^{n-1-v} d_i x^i + \sum_{i=n-v}^{n-1} d_i (x^{v+i-n} + x^{i-n}) \\ &= \left(\sum_{i=0}^{n-1-v} d_i x^i + \sum_{i=-v}^{-1} d_{n+i} x^i \right) + \sum_{i=0}^{v-1} d_{n+i-v} x^i \end{aligned} \quad (3.57)$$

and

$$\begin{aligned} A(x) &= \sum_{i=-v}^{n-1-v} a_{v+i} x^i = \sum_{i=0}^{n-1-v} a_{v+i} x^i + \sum_{i=-v}^{-1} a_{v+i} (x^{n+i} + x^{v+i}) \\ &= \left(\sum_{i=0}^{n-1-v} a_{v+i} x^i + \sum_{i=n-v}^{n-1} a_{v-n+i} x^i \right) + \sum_{i=0}^{v-1} a_i x^i \end{aligned} \quad (3.58)$$

It is easy to see that the conversion from one representation to the other needs v XOR gates and one T_X time delay due to the parallelism. Now consider two elements a , b represented in shifted polynomial basis. In order to calculate $c = ab \bmod f(x)$, first we have to perform a polynomial multiplication:

$$S(x) = A(x)B(x) = x^{-2v} \sum_{t=0}^{2n-2} s_t x^t = \sum_{t=-2v}^{2n-2-2v} s_{t+2v} x^t = r_- + r + r_+ \quad (3.59)$$

where

$$s_t = \sum_{\substack{i+j=t \\ 0 \leq i, j \leq n-1}} a_i b_j = \begin{cases} \sum_{i=0}^t a_i b_{t-i} & \text{for } 0 \leq t \leq n-1 \\ \sum_{i=t+1-n}^{n-1} a_i b_{t-i} & \text{for } n \leq t \leq 2n-2 \end{cases} \quad (3.60)$$

and

$$r = \sum_{t=-v}^{n-1-v} s_{t+2v} x^t \quad \text{and} \quad r_- = \sum_{t=-2v}^{-1-v} s_{t+2v} x^t \quad \text{and} \quad r_+ = \sum_{t=n-v}^{2(n-1-v)} s_{t+2v} x^t \quad (3.61)$$

The next step is to reduce r_+ and r_- using the following reduction formulae:

$$\begin{aligned} x^i &= x^{k+i-n} + x^{i-n} & \text{for } n-v \leq i \leq 2n-2-2v \\ x^i &= x^{n+i} + x^{k+i} & \text{for } -2v \leq i \leq -v-1 \end{aligned} \quad (3.62)$$

The reduction results would be:

$$\begin{aligned} r^- &= \sum_{t=-2v}^{-1-v} s_{t+2v} x^{n+t} + \sum_{t=-2v}^{-1-v} s_{t+2v} x^{k+t} \\ &= \sum_{t=n-2v}^{n-1-v} s_{t+2v-n} x^t + \sum_{t=k-2v}^{k-1-v} s_{t+2v-k} x^t \end{aligned} \quad (3.63)$$

and

$$\begin{aligned} r^+ &= \sum_{t=n-v}^{2n-2-2v} s_{t+2v} x^{t-(n-k)} + \sum_{t=n-v}^{2n-2-2v} s_{t+2v} x^{t-n} \\ &= \sum_{t=k-v}^{k+n-2-2v} s_{t+2v+n-k} x^t + \sum_{t=-v}^{n-2-2v} s_{t+n+2v} x^t \end{aligned} \quad (3.64)$$

And the final result would be:

$$\begin{aligned}
C(x) &= \sum_{t=-v}^{n-1-v} c_{v+t} x^t = r + r^- + r^+ \\
&= \sum_{t=-v}^{n-1-v} s_{t+2v} x^t \\
&\quad + \left(\sum_{t=n-2v}^{n-1-v} s_{t+2v-n} x^t + \sum_{t=k-2v}^{k-1-v} s_{t+2v-k} x^t \right) \\
&\quad + \left(\sum_{t=k-v}^{k+n-2-2v} s_{t+2v+n-k} x^t + \sum_{t=-v}^{n-2-2v} s_{t+n+2v} x^t \right)
\end{aligned} \tag{3.65}$$

It is proved in [6] that the total complexity of this procedure is:

$$\begin{cases} \#AND = n^2 \\ \#XOR = n^2 - 1 \\ Delay = T_A + (1 + \lceil \log_2 n \rceil) T_x \end{cases} \tag{3.66}$$

Following the work presented in [6], Fan and Hasan presented a new multiplier architecture for trinomials in [7]. This architecture was based on *Toeplitz matrix* concept combined bit KOA method which would result in sub-quadratic space complexities. First, we review toeplitz matrices.

A $n \times n$ matrix is of type *toeplitz* if for $1 \leq i, j \leq n-1$ we have: $m_{k,i} = m_{k-1,i-1}$. By this definition it is obvious that any $n \times n$ toeplitz matrix is determined just by $2n - 1$ elements of the first column and first row. Thus, for adding two toeplitz matrices, just $2n - 1$ additions are needed. An example of a 5×5 toeplitz matrix is shown in figure (3.3). Although the total number of entries in this matrix is 25, but we only have 9 distinct entries:

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 \\ a_5 & a_0 & a_1 & a_2 & a_3 \\ a_6 & a_5 & a_0 & a_1 & a_2 \\ a_7 & a_6 & a_5 & a_0 & a_1 \\ a_8 & a_7 & a_6 & a_5 & a_0 \end{bmatrix}$$

Figure 3.3 A 5×5 Toeplitz Matrix

Now assume that we have a $n \times n$ toeplitz matrix \mathbf{T} and a $n \times 1$ column vector \mathbf{V} and $n = 2^i$. Here we can split these matrixes as following:

$$\begin{aligned} \mathbf{T} &= \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_0 \\ \mathbf{T}_2 & \mathbf{T}_1 \end{pmatrix} \\ \mathbf{V} &= \begin{pmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \end{pmatrix} \end{aligned} \quad (3.67)$$

In the above expressions \mathbf{T}_0 , \mathbf{T}_1 and \mathbf{T}_2 are $\frac{n}{2} \times \frac{n}{2}$ matrices and \mathbf{V}_0 and \mathbf{V}_1 are $\frac{n}{2} \times 1$ column vectors. Now for calculating the multiplication \mathbf{TV} we can do as following:

$$\mathbf{TV} = \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_0 \\ \mathbf{T}_2 & \mathbf{T}_1 \end{pmatrix} \begin{pmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \end{pmatrix} = \begin{pmatrix} \mathbf{P}_0 + \mathbf{P}_2 \\ \mathbf{P}_1 + \mathbf{P}_2 \end{pmatrix} \quad (3.68)$$

in which

$$\begin{aligned} \mathbf{P}_0 &= (\mathbf{T}_0 + \mathbf{T}_1)\mathbf{V}_1 \\ \mathbf{P}_1 &= (\mathbf{T}_1 + \mathbf{T}_2)\mathbf{V}_0 \\ \mathbf{P}_2 &= \mathbf{T}_1(\mathbf{V}_0 + \mathbf{V}_1) \end{aligned} \quad (3.69)$$

Now using KOA we can recursively apply this algorithm in i steps and prepare the final result. The space complexity of this method would be:

$$\begin{cases} \#AND = n^{\log_2 3} \\ \#XOR = 5.5n^{\log_2 3} - 6n - 0.5 \end{cases} \quad (3.70)$$

Now for a finite field $GF(2^n)$ with $f(x) = x^n + x^k + 1$ as the irreducible polynomial, if we choose SPB with $v = k$ then the mastrovito matrix \mathbf{Z} can be changed into a toeplitz matrix by the following equation:

$$\mathbf{Z}' = \mathbf{UZ} \quad (3.71)$$

with

$$U = \begin{bmatrix} 0 & I_{(n-v)(n-v)} \\ I_{v \times v} & 0 \end{bmatrix} \quad (3.72)$$

Where $I_{v \times v}$ is the $v \times v$ identity matrix. It is important to note that this conversion is implemented by re-wiring only; therefore it doesn't add any gates to the space complexity of the multiplier.

3.2 Serial Finite Field Multipliers

One of the best serial finite field multipliers was presented in [1] by Berlekamp. In this multiplier, in order to find the product GZ in finite field $GF(2^n)$, G will be represented in polynomial basis whereas Z will be represented in its dual basis. Furthermore we assume that G is a constant and Z is a variable stored in a n -bit register. In the dual basis, the coefficients of GZ are the bits $Tr(GZx^i)$, for $0 \leq i \leq n-1$, which may be viewed as the following sequence:

$$\begin{aligned} & Tr(GZ) \\ & Tr(G(Zx)) \\ & \vdots \\ & Tr(G(Zx^{n-1})) \end{aligned} \quad (3.73)$$

It should be noted that, $Tr(GZ)$ is obtained by a single parity check on some subset of the bits of the Z register. Also, $Tr(G(Zx))$ can be obtained by the same parity check on the same register only by changing its contents from Z to xZ . The hardware architecture of such multiplier is shown in the next figure:

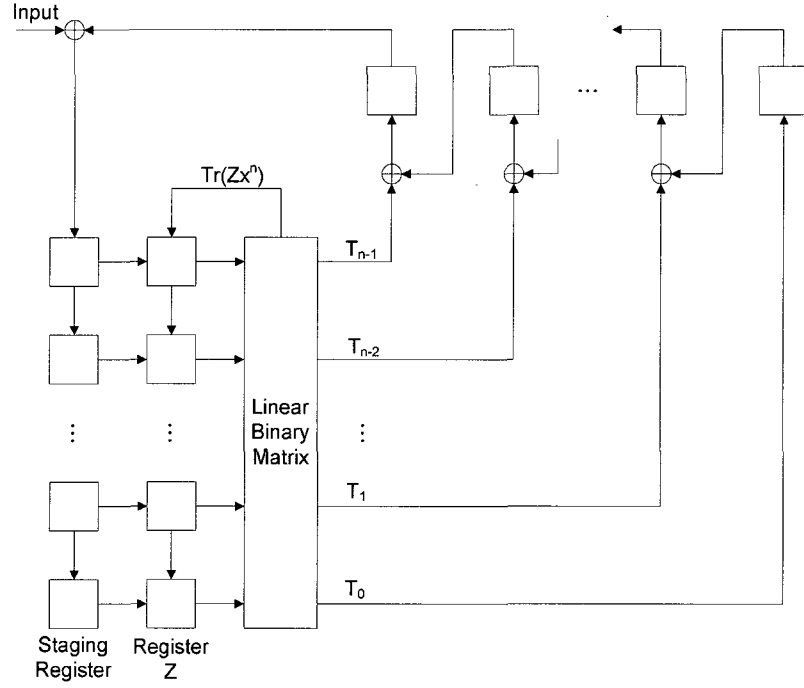


Figure 3.4 Berlekamp Multiplier

In the multiplier depicted in figure (3.4) the register feeding the linear binary matrix contains the value Z . So the outputs are:

$$\begin{aligned}
 T_0 &= Tr(g_0 Z) \\
 T_1 &= Tr(g_1 Z) \\
 &\vdots \\
 T_{n-2} &= Tr(g_{n-2} Z) \\
 T_{n-1} &= Tr(g_{n-1} Z)
 \end{aligned} \tag{3.74}$$

The output of the binary matrix which feeds back into Z is $tr(Zx^n)$. Since Z is in dual basis mode, next clock cycle multiplies Z by x . Now the outputs would be:

$$\begin{aligned}
T_0 &= Tr(g_0 x Z) \\
T_1 &= Tr(g_1 x Z) \\
&\vdots \\
T_{n-2} &= Tr(g_{n-2} x Z) \\
T_{n-1} &= Tr(g_{n-1} x Z)
\end{aligned} \tag{3.75}$$

And after n clock cycles we will have the final results which are:

$$\begin{aligned}
T_0 &= Tr(g_0 x^{n-1} Z) \\
T_1 &= Tr(g_1 x^{n-1} Z) \\
&\vdots \\
T_{n-2} &= Tr(g_{n-2} x^{n-1} Z) \\
T_{n-1} &= Tr(g_{n-1} x^{n-1} Z)
\end{aligned} \tag{3.76}$$

Now the multiplication is completed. This algorithm is very efficient in the sense that it requires minimum circuitry. However, the algorithm to multiply two elements of $GF(2^n)$ requires to represent one factor by a canonical basis and the other factor by the corresponding dual basis and the product is obtained in the dual basis, so proper base conversion is also needed.

Another good architecture for serial finite field multipliers was introduced by Massey and Omura in [20]. This multiplier uses normal basis for representing field elements. Consider finite field $GF(2^n)$ with normal basis $N = \{\alpha, \alpha^2, \alpha^{2^2}, \dots, \alpha^{2^{n-1}}\}$, with α be a root of the irreducible polynomial of the field. Any element a of the field is represented as: $a = a_0\alpha + a_1\alpha^2 + \dots + a_{n-1}\alpha^{2^{n-1}}$. Since in $GF(2^n)$ with normal basis we have $\alpha^{2^n} = \alpha$, squaring an element is equal to a cyclic shift operation:

$$\begin{aligned}
a^2 &= a_0\alpha^2 + a_1\alpha^{2^2} + \dots + a_{n-1}\alpha^{2^n} \\
&= a_{n-1}\alpha + a_0\alpha^2 + \dots + a_{n-2}\alpha^{2^{n-1}}
\end{aligned} \tag{3.77}$$

Now consider two elements of the field as a, b . The product of these two elements would be:

$$c = ab = [a_0, a_1, \dots, a_{n-1}][b_0, b_1, \dots, b_{n-1}] = [c_0, c_1, \dots, c_{n-1}] \quad (3.78)$$

The last term of the product; i.e. c_{n-1} , is some binary function of a, b :

$$c_{n-1} = f(a_0, a_1, \dots, a_{n-1}; b_0, b_1, \dots, b_{n-1}) \quad (3.79)$$

Now by squaring both sides of (3.78) we have:

$$c^2 = a^2 b^2 = [a_{n-1}, a_0, a_1, \dots, a_{n-2}][b_{n-1}, b_0, b_1, \dots, b_{n-2}] = [c_{n-1}, c_0, c_1, \dots, c_{n-2}] \quad (3.80)$$

Hence, the last component c_{n-2} of the product is obtained by the same function f operating on the components of a^2 and b^2 :

$$c_{n-2} = f(a_{n-1}, a_0, a_1, \dots, a_{n-2}; b_{n-1}, b_0, b_1, \dots, b_{n-2}) \quad (3.81)$$

By similarity we have:

$$\begin{cases} c_{n-1} = f(a_0, a_1, \dots, a_{n-1}; b_0, b_1, \dots, b_{n-1}) \\ c_{n-2} = f(a_{n-1}, a_0, a_1, \dots, a_{n-2}; b_{n-1}, b_0, b_1, \dots, b_{n-2}) \\ \vdots \\ c_1 = f(a_2, \dots, a_{n-1}, a_0, a_1; b_2, \dots, b_{n-1}, b_0, b_1) \\ c_0 = f(a_1, \dots, a_{n-1}, a_0; b_1, \dots, b_{n-1}, b_0) \end{cases} \quad (3.82)$$

The above equations define this multiplier. Figure (3.5) shows the block diagram of the multiplier:

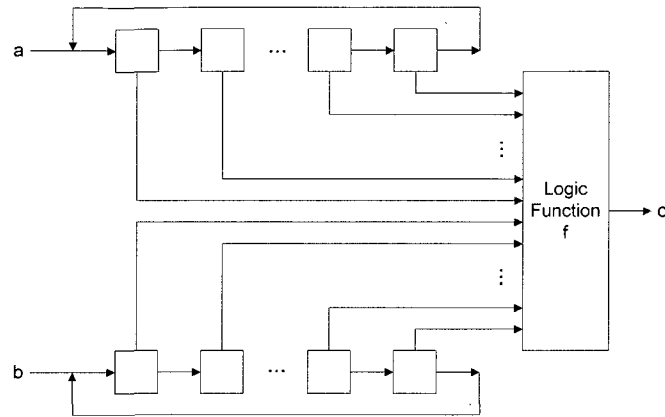


Figure 3.5 Massey-Omura Serial Multiplier

3.3 Summary

There are different types of finite field multipliers, with different advantages and disadvantages. In general we can categorize different finite field multipliers based on the hardware architecture of the multiplier into 3 types. Table (3.1) shows these types.

Hardware Architecture	Reference
Bit Parallel	[21], [27], [8], [28], [17], [37], [36], [9], [18], [41], [39], [40], [34], [2], [6], [7], [32]
Bit Serial	[1], [20], [12], [11]
Hybrid	[22], [26]

Table 3.1 Finite Field Multipliers Based on Hardware Architecture

One important factor in the architecture of finite field multipliers is the way of representing field elements in that multiplier. Some multipliers use the same representation in both inputs and output, while there are other multipliers which may have different representations for inputs and output. It is also important to consider the base conversion overhead in such multipliers. Table (3.2) shows some of these different types of multiplication methods:

Element Representation	Reference
Polynomial Basis	[21], [27], [28], [17], [37], [36], [9], [18], [41], [7], [32]
Normal Basis	[17], [20]
Dual Basis	[1], [8], [34]
Shifted Polynomial Basis	[6], [29]
Redundant Representation	[40], [2]
Weekly Dual Basis	[39]
Triangular Basis	[11]

Table 3.2 Finite Field Multipliers Based on Element Representation

In the context of this research, we are interested in bit parallel polynomial basis finite field multipliers. Parallel PB multipliers have three main types which are:

- Conventional
- Mastrovito
- Modified Mastrovito

There are also some other variations on these types, like the work presented in [17]. In these multipliers, the lowest complexities are gained when the polynomial of the field is an equally spaced trinomial. Any other trinomial is the next best choice along with all-one polynomials and other equally spaced trinomials.

Since we are interested in finite fields of form $GF(2^n)$, it is important to note that these efficient irreducible polynomials do not exist for many values of n . In such cases, the next best choice would be a pentanomial. Although the existence of an irreducible pentanomial for any value of n has not been proved yet, practically speaking, we can find at least one irreducible pentanomial for different values of n . The complexity of a multiplier based on an irreducible pentanomial highly depends on the form of the pentanomial, and we reviewed some of them in the previous sections.

Any other multiplier architecture for any type of irreducible polynomial should have better complexities than pentanomials. Table (3.3) shows the best results of different polynomial basis finite field multipliers in $GF(2^n)$:

Multiplier Type	Irreducible Polynomial	#AND	#XOR	Time Delay	Reference
Mastrovito	Equally Spaced Trinomial	n^2	$n^2 - \frac{n}{2}$	$T_A + (1 + \lceil \log_2 n \rceil)T_X$	[41]
Mastrovito	General ESP	n^2	$n^2 - \Delta$	$T_A + (1 + \lceil \log_2 n \rceil)T_X$	[41]
Mastrovito	General Trinomial	n^2	$n^2 - 1$	$T_A + (1 + \lceil \log_2 n \rceil)T_X$	[41]
Mastrovito Like	AOP	n^2	$n^2 - 1$	$T_A + (1 + \lceil \log_2(n-1) \rceil)T_X$	[17]
Conventional	Pentanomial	n^2	$n^2 + 2n - 3$	$T_A + (4 + \lceil \log_2(n-1) \rceil)T_X$	[32]

Table 3.3 Best Complexities of Different Finite Field Multipliers

It is obvious that any new proposed multiplier should have a complexity equal to or better than the complexity of pentanomial based multipliers presented in table (3.3).

4 New Finite Field Multiplier

4.1 Introduction

Finite field multipliers have application in many different areas such as cryptography, error correction codes, computer algebra, combinatorial designs and VLSI testing. Finite field multiplication has recently gained much attention due to its extensive use in public key cryptography and especially elliptic curve cryptosystems. In chapter 1, we pointed out the importance of research on finite field multipliers in cryptography systems. Research on different architectures of finite field multipliers mainly aim on reducing the space and time complexities. The goal of this thesis is to present a better finite field multiplier with smaller space and time complexities with regard to cryptographic applications.

As we mentioned before, finite field multipliers can be categorized into three major types based on their hardware architecture: *bit serial*, *bit parallel*, and *hybrid* multipliers.

For hardware implementation, serial multipliers are too slow because with the increasing key size of cryptographic applications, these multipliers need a considerable amount of clock pulses for encryption or decryption tasks.

On the other hand parallel multipliers are very fast since the input blocks enter the crypto-processor at the same time and in the next clock cycle the result is ready at the output ports. However these multipliers tend to occupy a large amount of silicon space on the micro-chip on which the hardware is fabricated.

Hybrid multipliers offer a balance between the above two types. By increasing the word size in these multipliers, the speed of multiplication increases. Decreasing the word size may result in less space on the chip. For each specific crypto-processor with a specific application and specific hardware limitations, a trade off between speed and space may be reached. But still bit parallel or, as called in some texts, full parallel multipliers are the fastest possible finite field multipliers.

Although hybrid multipliers seem to be the most suitable hardware architecture for physical implementation, full parallel multipliers are still in the center of attention and a huge amount of research is conducted on this subject. The reason for that is:

- In cryptographic applications speed is far more important than size
- Research on full parallel multipliers can also help in the parallel part of hybrid architectures
- With new VLSI technologies the amount of space needed for implementing certain circuits is reducing gradually

This is the main motivation behind our research.

Finite field multipliers can also be categorized based on the way they represent the elements of the field, or so called the basis of the field. Different basis multipliers were reviewed in chapter 3 and the pros and cons of each type was studied thoroughly. In this chapter, we are proposing a new polynomial basis full parallel finite field multiplier.

Parallel polynomial basis multipliers have two main types: conventional multiplier and mastrovito multiplier. There are also some variations on these two types. In this thesis a

new approach for multiplication is proposed. In this multiplier, we are using mastrovito multiplier concepts combined with the conventional method of multiplication. The result is not a mastrovito or conventional multiplier, but something in between. Depending on the type of the irreducible polynomial of the field, this approach may result in a finite field multiplier with better speed and space complexities.

We will present a full analysis of this proposed method along with a hardware architecture. Afterwards, we will apply the new multiplier to three newly proposed types of finite fields. By thoroughly examining the multiplier for these irreducible polynomials the exact amount of space and time complexities will be given. A comparison between our results and the best results of other finite field multipliers found in literature will show the advantage of our method.

4.2 Multiplier Architecture

We are proposing a new method for finite field multiplication. In this method we use both mastrovito multiplier and the conventional multiplier concepts. The first step in this method is to find a new way to build up the mastrovito matrix. We will do this by starting with the fundamental way of multiplying.

A finite field $GF(2^n)$ is defined as the set of residues modulo an irreducible polynomial $f(x)$ in which $f(x)$ is of degree n with the following form:

$$\begin{cases} f(x) = x^n + \sum_{i=1}^{n-1} f_i x^i + 1 \\ f_i \in GF(2) \end{cases} \quad (4.1)$$

Consider two elements of this field as: a, b . By employing polynomial basis we have:

$$\begin{cases} A(x) = \sum_{i=0}^{n-1} a_i x^i \\ B(x) = \sum_{i=0}^{n-1} b_i x^i \end{cases} \quad (4.2)$$

Polynomial multiplication of $A(x)$ and $B(x)$ would result in:

$$D(x) = A(x) \times B(x) = \sum_{i=0}^{2n-2} d_i x^i \quad (4.3)$$

We rewrite (2.14) formula as:

$$d_i = \begin{cases} \sum_{j=0}^i a_{i-j} b_j & \text{for } 0 \leq i \leq n-1 \\ \sum_{j=i-m+1}^{m-1} a_{i-j} b_j & \text{for } n \leq i \leq 2n-2 \end{cases} \quad (4.4)$$

From the above formula we have:

$$d_{n+k} = \sum_{j=k+1}^{n-1} a_{n+k-j} b_j \quad \text{for } 0 \leq k \leq n-2 \quad (4.5)$$

By substituting this formula in (2.22) we will have:

$$\begin{aligned} C(x) &= \sum_{i=0}^{n-1} d_i x^i + \sum_{i=0}^{n-1} \sum_{k=0}^{n-2} d_{n+k} t_{k,n-i-1} x^i \\ &= \sum_{i=0}^{n-1} \left(\sum_{j=0}^i a_{i-j} b_j \right) x^i + \sum_{i=0}^{n-1} \sum_{k=0}^{n-2} \left(\sum_{j=k+1}^{n-1} a_{n+k-j} b_j \right) t_{k,n-i-1} x^i \end{aligned} \quad (4.6)$$

It should be reminded that $t_{a,b}$ is the entry on the a 'th row and b 'th column of the transfer matrix \mathbf{T} . Transfer matrix is the matrix which converts powers of x^{n+k} for $0 \leq k \leq n-2$ to the sum of powers of x^i for $0 \leq i \leq n-1$:

By rewriting (4.6) we will have:

$$C(x) = \sum_{i=0}^{n-1} \left[\left(\sum_{j=0}^i a_{i-j} b_j \right) + \left(\sum_{k=0}^{n-2} \left(\sum_{j=k+1}^{n-1} a_{n+k-j} b_j \right) t_{k,n-i-1} \right) \right] x^i = \sum_{i=0}^{n-1} c_i x^i \quad (4.7)$$

So it is obvious that the coefficients of $C(x)$, the product of $A(x)$ and $B(x)$, are:

$$c_i = \sum_{j=0}^i a_{i-j} b_j + \sum_{k=0}^{n-2} \left(\sum_{j=k+1}^{n-1} a_{n+k-j} b_j \right) t_{k,n-i-1} \quad (4.8)$$

Now let's take a look at the matrix form of multiplication. As before we assume that an element a of a finite field of form $GF(2^n)$ in polynomial basis is represented as:

$$a = A(x) = \sum_{i=0}^n a_i x^i$$

The above formulation can be written as:

$$a = (a_0 \ a_1 \ \dots \ a_{n-2} \ a_{n-1}) (x^0 \ x^1 \ \dots \ x^{n-2} \ x^{n-1})^T = \mathbf{A}^T \mathbf{X} \quad (4.9)$$

Here \mathbf{A} is the coordinate column vector of the element, and \mathbf{X} is the vector of the basis. As we saw before in chapter 2, we can write the matrix form of multiplication as follows:

$$\mathbf{D} = \mathbf{M}\mathbf{B}$$

Here \mathbf{D} is the coordinate column vector of d , the product of a , b , \mathbf{B} is the coordinate column vector of b , and \mathbf{M} is the multiplication matrix which contains the elements of \mathbf{A} with the following form:

$$\mathbf{M}_{(2n-1)(n)} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & a_{n-4} & \cdots & a_0 & 0 \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_1 & a_0 \\ 0 & a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{n-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} & a_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1} \end{bmatrix} \quad (4.10)$$

Matrix \mathbf{M} can be divided into two matrices: The upper part, \mathbf{U} , and the lower part, \mathbf{L} , as follows:

$$\mathbf{M} = \begin{bmatrix} \mathbf{U} \\ \mathbf{L} \end{bmatrix} \quad (4.11)$$

where

$$\mathbf{U}_{n \times n} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 \\ a_1 & a_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{bmatrix} \quad (4.12)$$

and

$$L_{(n-1) \times n} = \begin{bmatrix} 0 & a_{n-1} & a_{n-2} & \cdots & a_1 \\ 0 & 0 & a_{n-1} & \cdots & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1} \end{bmatrix} \quad (4.13)$$

It should be noted that in thesis we use *Matlab* matrix notations to show column vectors and row vectors of a certain matrix, e.g., $\mathbf{Z}(i, :)$, $\mathbf{Z}(:, j)$ represent the i 'th row vector, and j 'th column vector in matrix \mathbf{Z} , respectively.

Considering \mathbf{U} and \mathbf{L} matrices we have:

$$\begin{aligned} \mathbf{U}(i, :) &= \left(a_i \ a_{i-1} \ \cdots \ a_1 \ a_0 \ \underbrace{0 \ 0 \ \cdots \ 0 \ 0}_{n-i-1} \right) \\ \mathbf{L}(i, :) &= \left(\underbrace{0 \ 0 \ \cdots \ 0 \ 0}_{i+1} \ a_{n-1} \ a_{n-2} \ \cdots \ a_{i+2} \ a_{i+1} \right) \end{aligned} \quad (4.14)$$

So if we multiply these vectors by the coordinate column vector of b we will have:

$$\begin{aligned} \mathbf{U}(i, :)\mathbf{B} &= \sum_{j=0}^i a_{i-j} b_j \\ \mathbf{L}(i, :)\mathbf{B} &= \sum_{j=i+1}^{n-1} a_{n-j+i} b_j \end{aligned} \quad (4.15)$$

We will use these matrices in our multiplier.

Now let's take a closer look at (4.8):

$$\begin{aligned}
c_i &= \sum_{j=0}^i a_{i-j}b_j + \sum_{k=0}^{n-2} \left(\sum_{j=k+1}^{n-1} a_{n+k-j}b_j \right) t_{k,n-1-i} \\
&= \sum_{j=0}^i a_{i-j}b_j + t_{0,n-1-i} \sum_{j=1}^{n-1} a_{n-j}b_j + t_{1,n-1-i} \sum_{j=2}^{n-1} a_{n-j+1}b_j + \dots \\
&\quad + t_{n-3,n-1-i} \sum_{j=n-2}^{n-1} a_{2n-j-3}b_j + t_{n-2,n-1-i} \sum_{j=n-1}^{n-1} a_{2n-j-2}b_j \\
&= \mathbf{U}(i, :) \mathbf{B} + t_{0,n-1-i} \mathbf{L}(0, :) \mathbf{B} + \dots + t_{n-2,n-1-i} \mathbf{L}(n-2, :) \mathbf{B}
\end{aligned} \tag{4.16}$$

In the above formulation we have a vector with a special form of:

$$t_{k,n-1-i} \mathbf{L}(k, :) \quad \text{for} \quad 0 \leq k \leq n-2 \tag{4.17}$$

Here we define a new set of matrices, called \mathbf{V} matrices, from \mathbf{V}_0 to \mathbf{V}_{n-2} . These $n \times n$ matrices are the key element of our new method. For each \mathbf{V}_k matrix, the i 'th row would be as follows:

$$\mathbf{V}_k(i, :) = t_{k,n-1-i} \mathbf{L}(k, :) \tag{4.18}$$

In other words, row i of matrix \mathbf{V}_k is row i of matrix \mathbf{L} if $t_{k,n-1-i} = 1$, otherwise it is a zero vector:

$$\mathbf{V}_k(i, :) = \begin{cases} \mathbf{L}(k, :) & \text{if } t_{k,n-1-i} = 1 \\ (0 \ 0 \ \dots \ 0) & \text{if } t_{k,n-1-i} = 0 \end{cases} \tag{4.19}$$

An example would clarify this better:

Consider finite field $GF(2^5)$ with $f(x) = x^5 + x^2 + 1$ as the field's irreducible polynomial.

Matrices \mathbf{L} and \mathbf{T} are shown below:

$$\mathbf{L} = \begin{bmatrix} 0 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & a_4 & a_3 \\ 0 & 0 & 0 & 0 & a_4 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Here we will have \mathbf{V}_0 , \mathbf{V}_1 , \mathbf{V}_2 , and \mathbf{V}_3 . In order to construct these matrices we look at \mathbf{L} and we refer to \mathbf{T} as our reference. Figure (4.1) shows these \mathbf{V} matrices:

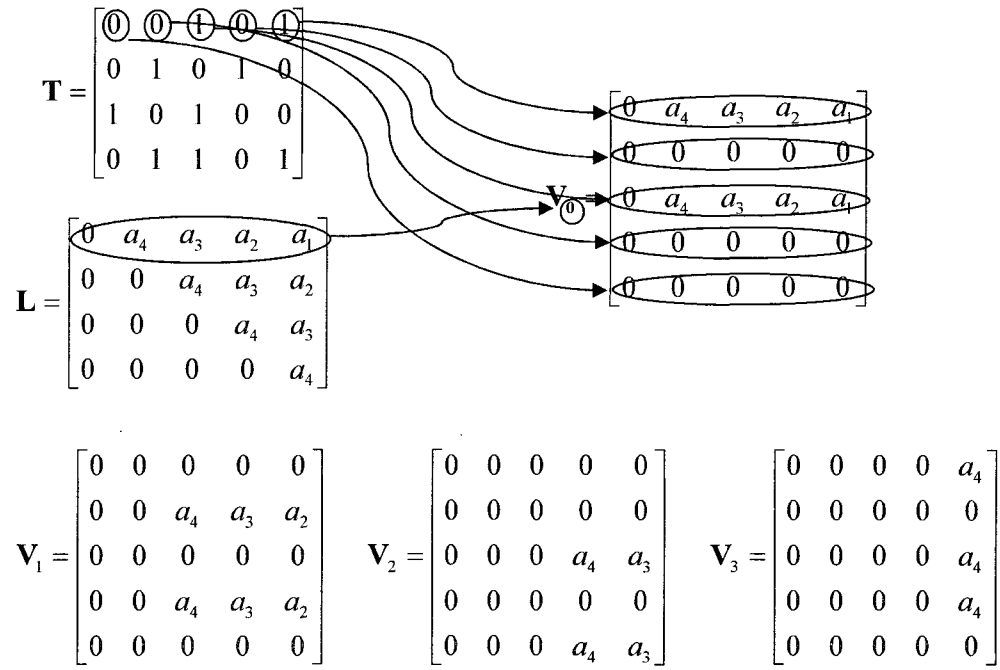


Figure 4.1 \mathbf{V} Matrices of $\text{GF}(2^5)$ with $f(x) = x^5 + x^2 + 1$

Using these \mathbf{V} matrices in (4.16) we have:

$$c_i = \mathbf{U}(i, :) \mathbf{B} + \mathbf{V}_0(i, :) \mathbf{B} + \mathbf{V}_1(i, :) \mathbf{B} + \cdots + \mathbf{V}_{n-2}(i, :) \mathbf{B} \quad (4.20)$$

This is the formulation for the i 'th coefficient of c , which is the i 'th entry in the coordinate column vector $\mathbf{C} = (c_0 \ c_1 \ \dots \ c_i \ \dots \ c_{n-1})$. Following the matrix multiplication rules we will have:

$$\begin{aligned}
\mathbf{C} &= \mathbf{UB} + \mathbf{V}_0\mathbf{B} + \mathbf{V}_1\mathbf{B} + \cdots + \mathbf{V}_{n-2}\mathbf{B} \\
&= (\mathbf{U} + \mathbf{V}_0 + \mathbf{V}_1 + \cdots + \mathbf{V}_{n-2})\mathbf{B}
\end{aligned} \tag{4.21}$$

Mastrovito multiplier was thoroughly reviewed in chapter 3. In the above formulation if we can consider $\mathbf{Z} = \mathbf{U} + \mathbf{V}_0 + \mathbf{V}_1 + \dots + \mathbf{V}_{n-2}$, then we have a form of mastrovito multiplier: $\mathbf{C} = \mathbf{ZB}$. So our goal of building mastrovito matrix is reached.

The first step in the new approach for building a finite field multiplier was to construct the mastrovito matrix, \mathbf{Z} , with matrices \mathbf{U} , and \mathbf{V}_0 to \mathbf{V}_{n-2} . The second step is to actually perform the above multiplication of \mathbf{ZB} and find the result. But in this method instead of performing just one matrix multiplication, \mathbf{ZB} , The elements of \mathbf{Z} , i.e. \mathbf{U} and \mathbf{V}_i s, are multiplied by \mathbf{B} individually and then the final result is gained by adding up these partial results. Figure (4.2) shows the hardware architecture of this multiplier.

In this hardware architecture, first the \mathbf{V} matrices are built with rewiring the input signal, a . In the next level, there is an AND page which performs the first part of matrix multiplication by performing AND operation between a , b . These AND gates are placed in parallel so all the AND operations are performed at once. The next level contains an XOR page in order to complete the matrix multiplication operation. These XOR gates are placed in a binary tree structure. Finally the last level contains some XOR gates as well. These XOR gates are to sum up the partial results and make the final output, c .

Space Complexity

As mentioned before in chapter 2, space complexity is expressed in terms of number of 2-input AND gates and 2-input XOR gates. AND gates are used in the first level and according to the structure of AND page in the hardware diagram, the total number of AND gates is calculated as following:

$$\begin{aligned}
\#AND &= 1 + 2 + \dots + n - 1 + n + n - 1 + \dots + 2 + 1 \\
&= \frac{n(n+1)}{2} + \frac{(n-1)n}{2} \\
&= n^2
\end{aligned} \tag{4.22}$$

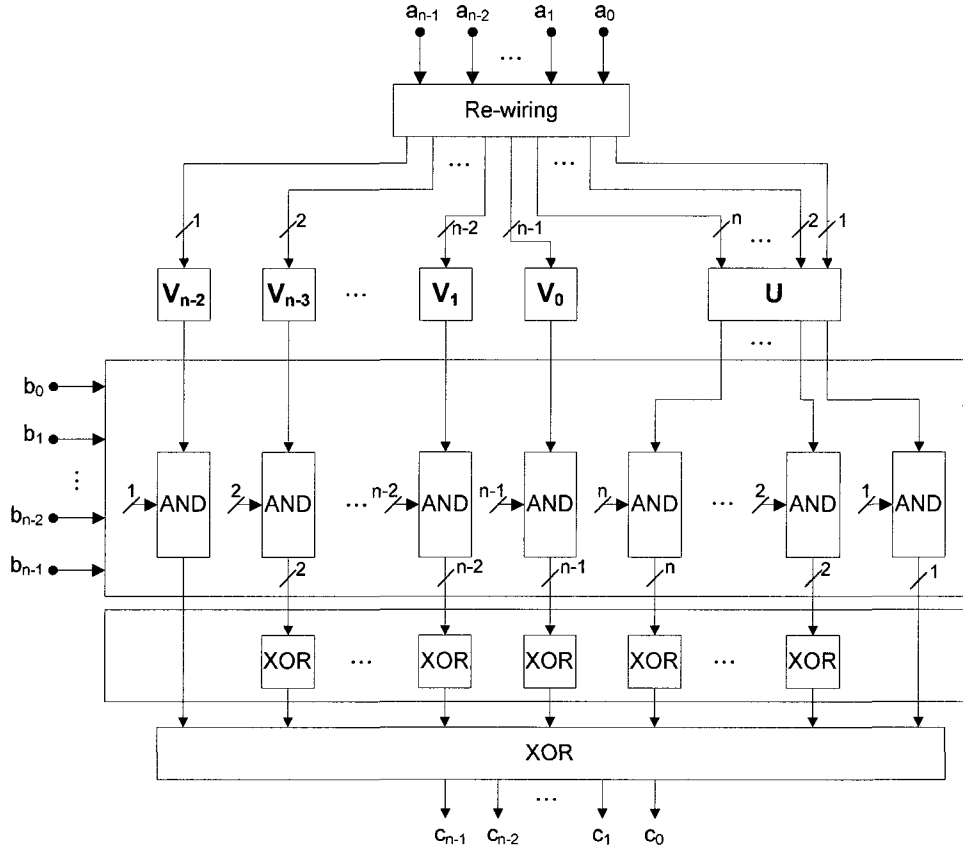


Figure 4.2 Proposed Multiplier Architecture

XOR gates are used in the two last levels of the hardware structure. The number of XOR gates in the last level of the hardware directly depends on the type of the irreducible polynomial of the field. The reason is that the number of summations in this level is determined by the shape of the V matrices which are based on the T matrix. So in this step we only calculate the number of XOR gates in the second level of hardware. Based on the hardware diagram we have:

$$\begin{aligned}
\#XOR &= 0 + 1 + \dots + n - 2 + n - 1 + n - 2 + \dots + 1 + 0 \\
&= \frac{(n-1)n}{2} + \frac{(n-2)(n-1)}{2} \\
&= n^2 - 2n + 1
\end{aligned} \tag{4.23}$$

If we represent the number of XOR gates in the last level by X_S the space complexity would be:

- Total number of AND gates : n^2
- Total number of XOR gates : $n^2 - 2n + 1 + X_S$

Time Complexity

Time complexity is always expressed in terms of the delay of 2-input AND gates, T_A , and the delay of 2-input XOR gates, T_X . In the first level, all the AND gates are placed in parallel so the delay of this level is T_A . In the second level by using a binary tree structure, the maximum delay would be $\lceil \log_2(n-1) \rceil T_X$. In the second level the accurate delay depends on the type of the irreducible polynomial of the field as well. If we denote this delay with xT_X the total time complexity would be:

- Time complexity : $T_A + (\lceil \log_2(n-1) \rceil + x)T_X$

4.3 Applying the New Method to Classes of Finite Fields

As we mentioned before, space and time complexity of our new method of finite field multiplication highly depends on the type of the irreducible polynomial of the field. In this section the results of applying the proposed method of multiplication on some classes of finite fields is examined thoroughly and the exact amount of space and time complexities are calculated.

4.3.1 Review of Three Classes of Finite Fields

Three classes of irreducible polynomials have been recently presented in [38]. These classes are called *Type I*, *Type II*, and *Type III*. These new types of finite fields are interesting because of their special form of T matrices. We will use these features when applying our new method of multiplication on these fields and we will show that the result are so efficient that can be used as an alternative for currently used finite field multipliers.

Type I Polynomials

Consider the finite field $GF(2^n)$, where $n \equiv 2 \pmod 3$ or in other words, $n = 3h + 2$ for some integer $h \geq 1$, and an irreducible polynomial $f(x)$ that defines $GF(2^n)$ be given in the following form:

$$f(x) = \sum_{j=0}^h (x^{3j+1} + x^{3j+2}) + 1 = \sum_{i=0}^{n-1} f_i x^i \quad (4.24)$$

$$\text{where } f_i = \begin{cases} 0 & \text{if } i \neq 0 \text{ and } i \not\equiv 0 \pmod 3 \\ 1 & \text{otherwise} \end{cases}$$

A good example would be for $h = 2$ and $n=8$. Here the polynomial would be:

$$f(x) = x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

In order to obtain the transfer matrix for an irreducible polynomial of type I, we have to solve x^{n+k} for $0 \leq k \leq n-2$. From (4.24) we have:

$$x^n = x^{3h+1} + \sum_{j=0}^{h-1} (x^{3j+1} + x^{3j+2}) + 1 \quad (4.25)$$

By multiplying x to both sides of the above equation we have:

$$\begin{aligned}
 x^{n+1} &= x^n + \sum_{j=0}^{h-1} (x^{3j+2} + x^{3j}) + x \\
 &= x^{3h+1} + \sum_{j=0}^{h-1} (x^{3j+1} + x^{3j+2}) + 1 + \sum_{j=0}^{h-1} (x^{3j+2} + x^{3j+3}) + x \quad (4.26) \\
 &= \sum_{j=1}^h (x^{3j} + x^{3j+1}) + 1
 \end{aligned}$$

It should be noted that in the above formulation, the operations on the coefficients are performed in $GF(2)$, i.e. addition is equal to subtraction. By continuing on multiplication of x to the both sides, the rest of the powers of x would be gained:

$$\begin{aligned}
 x^{n+2} &= \sum_{j=1}^h (x^{3j+1} + x^{3j+2}) + x \\
 &= x^n + x^{3h+1} + \sum_{j=1}^{h-1} (x^{3j+1} + x^{3j+2}) + x \quad (4.27) \\
 &= x^{3h+1} + \sum_{j=0}^{h-1} (x^{3j+1} + x^{3j+2}) + 1 + x^{3h+1} + \sum_{j=1}^{h-1} (x^{3j+1} + x^{3j+2}) + x \\
 &= x^2 + 1
 \end{aligned}$$

And generally

$$x^{n+k} = x^k + x^{k-2} \quad \text{for} \quad 2 \leq k \leq n-2 \quad (4.28)$$

According to the above equations, the transfer matrix of an irreducible polynomial of type I can be formed with the following properties:

1. Row $\mathbf{T}(0, :)$ has $2h + 2 = \frac{2}{3}(n + 1)$ '1's which are located at x^{3i+1} for $0 \leq i \leq h$, x^{3i+2} for $0 \leq i \leq h-1$, and x^0 . It should be noted that $\mathbf{T}(0, j)$ has a weight of x^{n-1-j} .
2. Row $\mathbf{T}(1, :)$ has $2h + 1 = \frac{2n-1}{3}$ '1's which are located at x^{3i} for $0 \leq i \leq h$, x^{3i+1} for $1 \leq i \leq h$.
3. Rows $\mathbf{T}(k, :)$ for $2 \leq k \leq n-2$ have just two '1's which are located at x^k and x^{k-2} .

The transfer matrix for the previous example is shown in figure (4.3).

$$\mathbf{T} = \begin{bmatrix} 1 & & 1 & 1 & & 1 & 1 & 1 \\ 1 & 1 & & 1 & 1 & & & 1 \\ & & & & & 1 & & 1 \\ & & & & 1 & & 1 & \\ & & & 1 & & 1 & & \\ & & 1 & & 1 & & & \\ & 1 & & 1 & & & & \end{bmatrix}$$

Figure 4.3 Transfer Matrix of a Type I Polynomial

Type II Polynomials

In this type of finite fields $n = 3h$ while h is an integer. The irreducible polynomial of this type of field, $f(x)$, has the following form:

$$f(x) = x^{3h} + \sum_{j=0}^{h-1} (x^{3j+2} + x^{3j}) = \sum_{i=0}^{n-1} f_i x^i \quad (4.29)$$

$$\text{where } f_i = \begin{cases} 0 & \text{if } i \equiv 1 \pmod{3} \\ 1 & \text{otherwise} \end{cases}$$

An example of this type of polynomials is for $h = 3$ and $n = 9$, resulting in $f(x)$ being:

$$f(x) = x^9 + x^8 + x^6 + x^5 + x^3 + x^2 + 1$$

In order to build the transfer matrix for this type of field, following the same procedure as before, x^{n+k} for $0 \leq k \leq n-2$ can be solved as follows:

$$\begin{cases} x^n = \sum_{j=0}^{h-1} (x^{3j+2} + x^{3j}) \\ x^{n+1} = \sum_{j=0}^{h-1} (x^{3j+2} + x^{3j+1}) + 1 \\ x^{n+k} = x^{k-1} + x^{k-2} \quad \text{for } 2 \leq k \leq n-2 \end{cases} \quad (4.30)$$

According to the above equations, the transfer matrix of an irreducible polynomial of type II can be formed with the following properties:

1. Row $\mathbf{T}(0, :)$ has $2h = \frac{2}{3}n$ '1's which are located at x^{3i} and x^{3i+2} for $0 \leq i \leq h-1$.
2. Row $\mathbf{T}(1, :)$ has $2h+1 = \frac{2n+3}{3}$ '1's which are located at x^{3i+1} and x^{3i+2} for $0 \leq i \leq h$, and x^0 .
3. Rows $\mathbf{T}(k, :)$ for $2 \leq k \leq n-2$ have just two '1's which are located at x^{k-1} and x^{k-2} .

The transfer matrix for the previous example is shown in figure (4.4):

$$\mathbf{T} = \begin{bmatrix} 1 & & 1 & 1 & & 1 & 1 & & 1 \\ 1 & 1 & & 1 & 1 & & 1 & 1 & 1 \\ & & & & & & & 1 & 1 \\ & & & & & & 1 & 1 & \\ & & & & & 1 & 1 & & \\ & & & & 1 & 1 & & & \\ & & & 1 & 1 & & & & \\ & & 1 & 1 & & & & & \\ & 1 & 1 & & & & & & \end{bmatrix}$$

Figure 4.4 T Matrix of a Type II Polynomial

Type III Polynomials

Consider the finite field $GF(2^n)$, where $n = 3h$ and h is a positive integer. Let the irreducible polynomial $f(x)$ that defines this field be given in the following form:

$$f(x) = x^{3h} + \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x + 1 = \sum_{i=0}^{n-1} f_i x^i \quad (4.31)$$

where $f_i = \begin{cases} 0 & \text{for } i = 2 \text{ or } i = 3j + 1 \text{ with } j > 1 \\ 1 & \text{otherwise} \end{cases}$

A good example for this type of polynomials would be for $h = 3$ and $n = 9$, and consequently $f(x)$ would be:

$$f(x) = x^9 + x^8 + x^6 + x^5 + x^3 + x + 1$$

Again x^{n+k} for $0 \leq k \leq n-2$ can be solved as follows:

$$x^n = \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x + 1 \quad (4.33)$$

By multiplying x we have:

$$\begin{aligned} x^{n+1} &= \sum_{j=1}^{h-1} (x^{3j+3} + x^{3j+1}) + x^2 + x \\ &= x^n + \sum_{j=1}^{h-1} (x^{3j} + x^{3j+1}) + x^3 + x^2 + x \\ &= \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x + 1 + \sum_{j=1}^{h-1} (x^{3j} + x^{3j+1}) + x^3 + x^2 + x \\ &= \sum_{j=1}^{h-1} (x^{3j+1} + x^{3j+2}) + x^3 + x^2 + 1 \end{aligned} \quad (4.34)$$

and

$$\begin{aligned}
x^{n+2} &= \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j+3}) + x^4 + x^3 + x \\
&= x^n + \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x^4 + x \\
&= \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x + 1 + \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x^4 + x \\
&= x^4 + 1
\end{aligned} \tag{4.35}$$

And generally:

$$x^{n+k} = x^{k+2} + x^{k-2} \quad \text{for} \quad 2 \leq k \leq n-3 \tag{4.36}$$

And finally the last one:

$$\begin{aligned}
x^{2n-2} &= x^n + x^{n-4} \\
&= \sum_{j=1}^{h-1} (x^{3j+2} + x^{3j}) + x + 1 + x^{3h-4} \\
&= x^{3h-1} + x^{3h-3} + x^{3h-4} + x^{3h-6} + \sum_{j=1}^{h-3} (x^{3j+2} + x^{3j}) + x + 1 + x^{3h-4} \\
&= x^{3h-1} + x^{3h-3} + x^{3h-6} + \sum_{j=1}^{h-3} (x^{3j+2} + x^{3j}) + x + 1
\end{aligned} \tag{4.37}$$

According to these equations, we can build up the transfer matrix of an irreducible polynomial of type III with the following properties:

1. Row $\mathbf{T}(0, :)$ has $2h = \frac{2}{3}n$ '1's which are located at x^{3i} for $0 \leq i \leq h-1$, x^{3i+2} for $1 \leq i \leq h-1$, and x^1 .

2. Row $\mathbf{T}(1, :)$ has $2h + 1 = \frac{2n+3}{3}$ '1's which are located at x^{3i+1} for $1 \leq i \leq h-1$, x^{3i+2} for $0 \leq i \leq h-1$, x^3 , and x^0 .
3. Rows $\mathbf{T}(k, :)$ for $2 \leq k \leq n-3$ have just two '1's which are located at x^{k+2} and x^{k-2} .
4. Row $\mathbf{T}(n-2, :)$ has $2h - 1 = \frac{2n-3}{3}$ '1's which are located at x^{3i} for $0 \leq i \leq h-1$, x^{3i+2} for $1 \leq i \leq h-3$, x^{3h-1} , and x^1 .

The transfer matrix of the previous example is shown in figure (4.5).

$$\mathbf{T} = \begin{bmatrix} 1 & & 1 & 1 & & 1 & & 1 & 1 \\ 1 & 1 & & 1 & 1 & 1 & 1 & & 1 \\ & & & & 1 & & & & 1 \\ & & & 1 & & & & 1 & \\ & & 1 & & & & 1 & & \\ & 1 & & & & 1 & & & \\ 1 & & & & 1 & & & & \\ 1 & & 1 & & & 1 & & 1 & 1 \end{bmatrix}$$

Figure 4.5 T Matrix of a Type III Polynomial

4.3.2 Results Related to Type I Polynomials

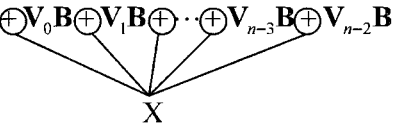
Before computing the results and finding the complexities, it is very important to build the \mathbf{V} matrices. As mentioned before, the \mathbf{T} matrix of a polynomial of type I has a special form. The first row of the transfer matrix, $\mathbf{T}(0, :)$, has $2h + 2$ '1's so \mathbf{V}_0 will have $2h + 2$ non-zero rows and all of these rows are the first row of Matrix \mathbf{L} , $\mathbf{L}(0, :)$; therefore they have $n - 1$ non-zero items. The locations of these rows are $\mathbf{V}_0(0, :)$, $\mathbf{V}_0(3i+1, :)$ and $\mathbf{V}_0(3i+2, :)$ for $0 \leq i \leq h-1$, and finally $\mathbf{V}_0(n-1, :)$.

The second row of \mathbf{T} matrix, $\mathbf{T}(1, :)$, has $2h + 1$ '1's so \mathbf{V}_1 will have the second row of \mathbf{L} , $\mathbf{L}(1, :)$ in $2h + 1$ rows and the rest of rows are all zero. Locations of the non-zero rows are $\mathbf{V}_1(0, :)$, $\mathbf{V}_1(3i, :)$ and $\mathbf{V}_1(3i+1, :)$ for $1 \leq i \leq h$. The rest of \mathbf{V} matrices, i.e. \mathbf{V}_k for $2 \leq k \leq n - 2$ only contain two non-zero rows which are $\mathbf{L}(k, :)$. These rows contain $n - 1 - k$ non-zero items.

Calculation of Space Complexity means to count the number of all 2-input AND gate and 2-input XOR gates needed to perform the multiplication. In the previous sections we found that the space complexity of this multiplier is:

$$\begin{cases} \#AND = n^2 \\ \#XOR = n^2 - 2n + 1 + X_S \end{cases} \quad (4.38)$$

At this point only X_S needs to be calculated. X_S corresponds to the XOR gates used to add up the partial results of matrix multiplications:

$$C = UB \oplus V_0 B \oplus V_1 B \oplus \dots \oplus V_{n-3} B \oplus V_{n-2} B$$


Since V_0 has $2h + 2$ nonzero rows the first addition needs $2h + 2$ XOR gates. Similarly, the second addition needs $2h + 1$ XOR gates and the rest of additions, each need two XOR gates each. So X_S is calculated as follows:

$$\begin{aligned} X_S &= (2h + 2) + (2h + 1) + \underbrace{2 + 2 + \dots + 2}_{n-3} \\ &= 2n - 6 + 4h + 3 \\ &= 2n - 3 + \frac{4}{2}n - \frac{8}{3} \end{aligned} \quad (4.39)$$

And the total number of XOR gates would be:

$$\begin{aligned} \#XOR &= n^2 - 2n + 1 + 2n - 3 - \frac{4}{2}n - \frac{8}{3} \\ &= n^2 + \frac{4}{3}n - \frac{14}{3} \end{aligned} \quad (4.40)$$

It is important to note that the XOR complexity can be reduced by eliminating the redundant terms. Looking at the V_0 and V_1 matrices, it can be seen that both of them have

a value on row 0, row $3i + 1$ for $1 \leq i \leq h-1$, and row $n - 1$. So in performing $\mathbf{V}_0\mathbf{B} + \mathbf{V}_1\mathbf{B}$ the term $\mathbf{L}(0, :) \mathbf{B} + \mathbf{L}(1, :) \mathbf{B}$ is repeated $h + 1$ times and only one of these summations is necessary, therefore h XOR gates will be eliminated:

$$\begin{aligned} \#XOR &= n^2 + \frac{4}{3}n - \frac{14}{3} - h \\ &= n^2 + \frac{4}{3}n - \frac{14}{3} - \frac{n}{2} + \frac{2}{3} \\ &= n^2 + n - 4 \end{aligned} \quad (4.41)$$

In the previous sections we found that the time complexity of this multiplier is:

$$T_A + ([\log_2(n - 1)] + x)T_X \quad (4.42)$$

in which xT_X is the amount of delay caused by the last level of XOR gates which perform the summation of partial multiplication results. In order to calculate the maximum value of x , the longest signal path should be determined. According to the **T** matrix of type I, the maximum number of '1's in one column is 4, so when performing the summation $\mathbf{UB} + \mathbf{V}_0\mathbf{B} + \dots + \mathbf{V}_{n-2}\mathbf{B}$ the longest signal path would have four XOR gates and according to the **T** matrix it would be at $\mathbf{U}(3i+1, :) \mathbf{B} + \mathbf{L}(0, :) \mathbf{B} + \mathbf{L}(1, :) \mathbf{B} + \mathbf{L}(3i+1, :) \mathbf{B} + \mathbf{L}(3i+3) \mathbf{B}$ for $1 \leq i \leq h - 1$.

It is important to note that these are the exact locations of the redundant term $\mathbf{L}(0, :) \mathbf{B} + \mathbf{L}(1, :) \mathbf{B}$. The biggest time delay will happen when $i = 1$. So the time delay of $\mathbf{U}(4, :) \mathbf{B} + \mathbf{L}(0, :) \mathbf{B} + \mathbf{L}(1, :) \mathbf{B} + \mathbf{L}(4, :) \mathbf{B} + \mathbf{L}(6, :) \mathbf{B}$ is the longest delay of the circuit. First let's see how much time is needed for the elements of the above summation to become ready.

Here $\mathbf{L}(0, :) \mathbf{B}$ needs $[\log_2(n - 1)]T_X$ time delay where T_X is the delay of a 2-input XOR gate. Time delays of the other terms are:

$$\begin{aligned} \mathbf{U}(4, :) \mathbf{B} &: [\log_2 5]T_X \\ \mathbf{L}(1, :) \mathbf{B} &: [\log_2(n - 2)]T_X \\ \mathbf{L}(4, :) \mathbf{B} &: [\log_2(n - 5)]T_X \\ \mathbf{L}(6, :) \mathbf{B} &: [\log_2(n - 7)]T_X \end{aligned}$$

Since $\lceil \log_2(n-1) \rceil = \lceil \log_2(n-2) \rceil = \lceil \log_2(n-5) \rceil = \lceil \log_2(n-7) \rceil$ for $n \geq 23$, we can assume that all of these five elements become ready at the same time. Therefore by implementing a binary tree structure the total delay would be $3T_X$:

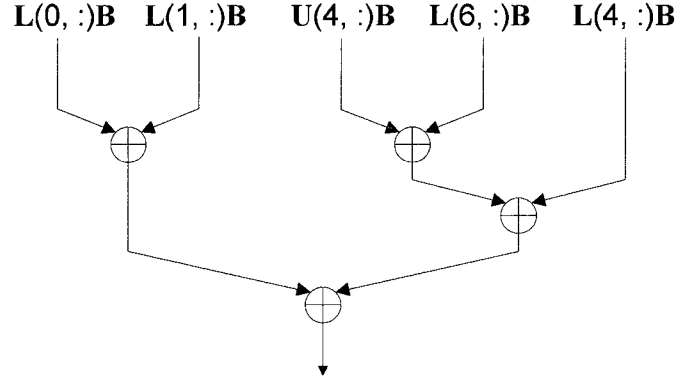


Figure 4.6 Longest Signal Pass of a Type I Multiplier

It is important to note that for $n < 23$, since $\lceil \log_2(n-7) \rceil < \lceil \log_2(n-1) \rceil$, we can perform the addition $U(4, :)B + L(6, :)B$ before the other terms become ready and eliminate one T_X delay. Table (4.1) shows the final complexities of our multiplier for type I polynomials:

#AND	#XOR	DELAY
n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_X$

Table 4.1 Complexity Results of a Type I Multiplier

4.3.3 Results Related to Type II Polynomials

The same technique would be used in calculating the results for Type II polynomials. According to the form of \mathbf{T} matrix for this type, the only difference would be in the form of \mathbf{V}_0 and \mathbf{V}_1 . Here \mathbf{V}_0 has $2h$ non-zero rows and they are row $\mathbf{V}_0(3i, :)$ and row $\mathbf{V}_0(3i+2, :)$ for $0 \leq i \leq h-1$. Also \mathbf{V}_1 will have $2h+1$ nonzero rows which are row

$V_1(0, :)$, row $V_1(3i+1, :)$, and row $V_1(3i+2, :)$ for $0 \leq i \leq h-1$. So X_S would be calculated as follows:

$$\begin{aligned}
 X_S &= (2h) + (2h+1) + \underbrace{2+2+\dots+2}_{n-3} \\
 &= 2n - 6 + 4h + 1 \\
 &= 2n - 5 + \frac{4}{3}n
 \end{aligned} \tag{4.43}$$

Here again the redundant terms would be found when calculating $V_0\mathbf{B} + V_1\mathbf{B}$ and the term $L_0\mathbf{B} + L_1\mathbf{B}$ is repeated $h+1$ times. So the XOR complexity would be:

$$\begin{aligned}
 \#XOR &= n^2 - 2n + 1 + 2n - 5 + \frac{4}{3}n - h \\
 &= n^2 + \frac{4}{3}n - 4 - \frac{n}{3} \\
 &= n^2 + n - 4
 \end{aligned} \tag{4.44}$$

The same thing will happen when calculating the time delay. For this polynomial the longest signal pass contains four XOR gates and according to the transfer matrix of this type, it happens when calculating $U(3i-1, :)\mathbf{B} + L(0, :)\mathbf{B} + L(1, :)\mathbf{B} + L(3i, :)\mathbf{B} + L(3i+1, :)\mathbf{B}$ for $1 \leq i \leq h-1$, and the biggest delay is when $i=1$ and $U(2, :)\mathbf{B} + L(0, :)\mathbf{B} + L(1, :)\mathbf{B} + L(3, :)\mathbf{B} + L(4, :)\mathbf{B}$ is being calculated. Again the result would be $(\lceil \log_2(n-1) \rceil + 3)T_x$ for $n \geq 15$ and $(\lceil \log_2(n-1) \rceil + 2)T_x$ for $n < 15$. Table (4.2) shows the final complexities of our multiplier for type II polynomials:

#AND	#XOR	DELAY
n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_x$

Table 4.2 Complexity Results of a Type II Multiplier

4.3.4 Results Related to Type III Polynomials

According to the special form of the \mathbf{T} matrix in this type, \mathbf{V}_0 has $2h$ non-zero rows which are: row $\mathbf{V}_0(1, :)$, row $\mathbf{V}_0(3i, :)$ for $0 \leq i \leq h-1$ and row $\mathbf{V}_0(3i+2, :)$ for $1 \leq i \leq h-1$. Likewise \mathbf{V}_1 has $2h+1$ nonzero rows which are: row $\mathbf{V}_1(0, :)$, row $\mathbf{V}_1(3, :)$, row $\mathbf{V}_1(3i+1, :)$ for $1 \leq i \leq h-1$, and row $\mathbf{V}_1(3i+2, :)$ for $0 \leq i \leq h-1$. All the other \mathbf{V} matrices have just two non-zero rows except for the last one \mathbf{V}_{n-2} which has $2h-1$ non-zero rows and they are: row $\mathbf{V}_{n-2}(0, :)$, row $\mathbf{V}_{n-2}(3i, :)$ for $0 \leq i \leq h-1$, row $\mathbf{V}_{n-2}(3i+2, :)$ for $1 \leq i \leq h-3$, and row $\mathbf{V}_{n-2}(h-1, :)$. Calculating X_S would be as follows:

$$\begin{aligned} X_S &= (2h) + (2h+1) + (2h-1) + \underbrace{2+2+\dots+2}_{n-4} \\ &= 2n - 8 + 6h \\ &= 4n - 8 \end{aligned} \tag{4.45}$$

Again many redundancies can be found in performing matrix multiplication. First considering \mathbf{V}_0 and \mathbf{V}_{n-2} you find that the term $\mathbf{L}(0, :)\mathbf{B} + \mathbf{L}(n-1, :)\mathbf{B}$ is repeated $2h-1$ times so $2h-2$ XOR gates can be eliminated. Next, considering \mathbf{V}_0 , \mathbf{V}_1 and \mathbf{V}_{n-2} it can be seen that the term $\mathbf{L}(0, :)\mathbf{B} + \mathbf{L}(1, :)\mathbf{B} + \mathbf{L}(n-1, :)$ is repeated h times, so again $h-1$ XOR gates will be eliminated and the final complexity would be:

$$\begin{aligned} \#XOR &= n^2 - 2n + 1 + 4n - 8 - (2h-2) - (h-1) \\ &= n^2 + 2n - 4 - 3h \\ &= n^2 + n - 4 \end{aligned} \tag{4.46}$$

Considering the form of \mathbf{T} matrix is important in calculating time delay. Since each column in \mathbf{T} matrix has maximum five '1's (for $n < 15$ it has maximum four '1's) it can be assumed that the longest signal path has five XOR gates. The longest delay will happen when calculating $\mathbf{U}(5, :)\mathbf{B} + \mathbf{L}(0, :)\mathbf{B} + \mathbf{L}(1, :)\mathbf{B} + \mathbf{L}(3, :)\mathbf{B} + \mathbf{L}(7, :)\mathbf{B} + \mathbf{L}(n-2, :)\mathbf{B}$ for $n > 15$. Using the same technique, the time complexity would be

$(\lceil \log_2(n-1) \rceil + 3)T_x$. Table (4.3) shows the final complexities of our multiplier for type III polynomials:

#AND	#XOR	DELAY
n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_x$

Table 4.3 Complexity Results of a Type III Multiplier

4.4 Comparisons

In this section we introduced a new method for building parallel finite field multipliers in polynomial basis. Then we applied our method to some types of fields presented in [38] and showed the complexity results. As mentioned in chapter 3, it is not possible to beat the results of a finite field multiplier built for a trinomial, ESP, or AOP. But since these types of polynomials do not exist for all finite fields that we deal with, it is important to find new multipliers that have better complexity results than pentanomials. Table (4.4) compares the results of our multiplier with the best results of a pentanomial based multiplier:

Type of Multiplier	Polynomial of the Field	#AND	#XOR	Delay
Conventional	Pentanomial	n^2	$n^2 + 2n - 3$	$T_A + (\lceil \log_2(n-1) \rceil + 4)T_x$
Proposed	Type I	n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_x$
Proposed	Type II	n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_x$
Proposed	Type II	n^2	$n^2 + n - 4$	$T_A + (\lceil \log_2(n-1) \rceil + 3)T_x$

Table 4.4 The Results of the New Multiplier vs the Best Pentanomial Multiplier

It is clearly seen that the results of our multiplier is better than a pentanomial based multiplier. Therefore this new multiplier can be used as an alternative for finite field multipliers based on pentanomials for finite fields of the form $GF(2^n)$ where an irreducible trinomial or ESP or AOP doesn't exist.

5 Conclusion

5.1 Summary of contributions

This work presents a new bit parallel polynomial basis finite field multiplier. Polynomial basis multipliers have two major types: conventional and Mastrovito. This new approach combines the aspects of the both types to achieve a more efficient architecture. By efficiency we mean both space and speed.

Among the different finite field multipliers in literature, those built for irreducible trinomials or equally spaced polynomials or all-one polynomials have the best space and time complexities. But for a finite field $GF(2^n)$, the problem is that there is no guarantee to find such an irreducible polynomial for any value of n . Currently when those optimized irreducible polynomials do not exist in a certain field, the multiplier is built based on a pentanomial.

The proposed multiplier has been applied to some classes of finite fields and resulted in very good complexities. It has smaller space complexity compared to the smallest finite field multiplier based on a pentanomial. At the same time it had smaller time complexity as well. So the efficiency of multiplication was improved for both space and speed. Therefore it can be a good alternative for polynomial based multipliers in the finite fields that an optimized irreducible polynomial does not exist.

5.2 Future Work

In this work only an architecture level of the multiplier was presented and the complexities were expressed in terms of some nominal values. The next step could be implementation of the multiplier as a real circuit for example in an FPGA in order to analyze the actual performance of it. It should also be considered that the full parallel multiplier architectures are impractical for circuit implementation when the value of n is big, therefore in the FPGA implementation a small value of n should be chosen. However, big values of n can also be used for simulation purposes.

Considering the architecture level, this method has been applied to only a few classes of finite field. Based on the good results of these fields, we can assume that some other classes of fields may exist with promising results as well. So the next step would be applying this method to these classes of finite fields. Furthermore, the matrix multiplication operations in this method can be combined with some efficient algorithms such as KOA to gain more reductions in complexities. The result of applying this new method on previously investigated polynomials such as trinomials, ESPs, or AOPs should also be considered in order to find any possible common structures.

References

- [1] E.R. Berlekamp, “Bit-Serial Reed-Solomon Encoders”, *IEEE Trans. Information Theory*, Vol. 28, pp. 869-874, Nov. 1982.
- [2] K. Chang, D. Hong, H. Cho, “Low Complexity Bit-Parallel Multiplier for $GF(2^m)$ Defined by All-One Polynomials Using Redundant Representation”, *IEEE Trans. Computers*, Vol.54, No. 12, pp. 1628-1630, Dec. 2005.
- [3] W. Diffie and M.E. Hellman, “New Directions in Cryptography”, *IEEE Trans. Information Theory*, Vol. 22, No. 6, pp. 644-654, Nov. 1976.
- [4] Taher El-Gamal, “A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”, *IEEE Trans. Information Theory*, Vol. IT-31, No. 4, pp.469–472, 1985.
- [5] M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Blumel, “A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^n)$ ”, *Proc. Cryptographic Hardware and Embedded Systems (CHES '02)*, pp. 381-399, 2003.
- [6] H. Fan, Y. Dai, “Fast Bit-Parallel $GF(2^n)$ Multiplier for All Trinomials”, *IEEE Trans. Computers*, Vol. 54, No. 4, pp. 485-490, Apr. 2005.
- [7] H. Fan, M. Hasan, “A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields”, *IEEE Trans. Computers*, Vol. 56, No. 2, pp. 224-233, Feb. 2007.

- [8] S.T.J. Fenn, M. Benaissa, and D. Taylor, "GF(2^m) Multiplication and Division over the Dual Basis", *IEEE Trans. Computers*, Vol. 45, No. 3, pp. 319-327, Mar 1996.
- [9] A. Halbutogullari, C. Koc, "Mastrovito Multiplier for General Irreducible Polynomials", *IEEE Trans. Computers*, Vol. 49, No. 5, pp. 503-518, May 2000.
- [10] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, Dec. 2003.
- [11] M.A. Hasan and V.K. Bhargava, "Architecture for Low Complexity Rate-Adaptive Reed-Solomon Encoder", *IEEE Trans. Computers*, Vol. 44, No. 7, pp. 938-942, Jul. 1995.
- [12] M.A. Hasan and V.K. Bhargava, "Division and Bit-Serial Multiplication over GF(2^m)", *IEE Proc.-E*, Vol. 139, No. 3, pp. 230-236, May 1992.
- [13] T. Itoh and S. Tsujii, "Structure of Parallel Multipliers for a Class of Fields GF(2^m)", *Information and Computation*, Vol. 83, pp. 21-40, 1989.
- [14] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata", *Soviet Physics-Doklady (English translation)*, Vol. 7, No. 7, pp. 595-596, 1963.
- [15] D.E. Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, 1998.
- [16] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, Vol. 48, pp. 203-209, 1987.

- [17] C.K. Koc and B. Sunar, "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields", *IEEE Trans. Computers*, Vol. 47, No. 3, pp. 353-356, Mar. 1998.
- [18] M. Leone, "A New Low Complexity Parallel Multiplier for a Class of Finite Fields", *Proc. Cryptographic Hardware and Embedded Systems (CHES 2001)*, pp. 160-170, 2001.
- [19] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Application*, Cambridge University Press, 1986.
- [20] J.L. Massey and J.K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US patent 4,587,627, 1986.
- [21] E.D. Mastrovito, "VLSI Architectures for Multiplication Over Finite Field $GF(2^m)$ ", *Proc. Sixth Int'l Conf., AAECC-6, T. Mora, ed.*, pp. 297-309, Rome, Jul. 1988.
- [22] P.K. Meher, Y. Ha, C.Y. Lee, "An Optimized Design for Serial-Parallel Finite Field Multiplication over $GF(2^m)$ Based on All-One Polynomials", *Proc. 2009 Conf. on Asia and South Pacific Design Automation*, Yokohama, Japan, pp. 210-215, 2009.
- [23] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
- [24] V.S. Miller, "Use of Elliptic Curves in Cryptography", *Advances in Cryptography Proc. Crypto '85*, pp. 417-426, 1985.

- [25] M. Morii, M. Kasahara, and D.L. Whiting, "Efficient Bit-Serial Multiplication and Discrete-Time Wiener-Hopf Equation Over Finite Fields," *IEEE Trans. Information Theory*, Vol. 35, pp. 1177-1184, 1989.
- [26] A.H. Namin, H. Wu, M. Ahmadi, "Comb Architectures for Finite Field Multiplication in F_2^m ", *IEEE Trans. Computers*, Vol. 56, No. 7, pp. 909-916, Jul. 2007.
- [27] C. Paar, "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields", *IEEE Trans. Computers*, Vol. 45, No. 7, pp. 856-861, Jul. 1996.
- [28] C. Paar, P. Fleischmann, and P. Roelse, "Efficient Multiplier Schemes for Galois Fields $GF(2^{4n})$ ", *IEEE Trans. Computers*, Vol. 47, No. 2, pp. 162-170, Feb. 1998.
- [29] S. Park, K. Chang, D. Hong, "Efficient Bit-Parallel Multiplier for Irreducible Pentanomials Using a Shifted Polynomial Basis", *IEEE Trans. Computers*, Vol. 55, No. 9, pp. 1211-1215, Sep. 2006.
- [30] O. Pretzel, *Error-Correcting Codes and Finite Fields*, Oxford University Press, 1996.
- [31] A. Reyhani-Masoleh, M. Hasan, "A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$ ", *IEEE Trans. Computers*, Vol. 51, No. 5, pp. 511-520, May. 2002.
- [32] A. Reyhani-Masoleh, M. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$ ", *IEEE Trans. Computers*, Vol. 53, No. 8, pp. 945-959, Aug. 2004.

- [33] R.L. Rivest, A. Shamir, and L.A. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Comm. ACM*, Vol. 21, pp. 120-126, 1978.
- [34] F. Rodriguez-Henriquez, C. Koc, "Parallel Multipliers Based on Special Irreducible Pentanomials", *IEEE Trans. Computers*, Vol. 52, No. 12, pp 1535-1542, Dec. 2003.
- [35] W. Stallings, *Cryptography and Network Security*, Prentice Hall, 2003.
- [36] L. Song, K. Parhi, "Low-Complexity Modified Mastrovito Multipliers Over Finite Fields $GF(2^m)$ ", *Proc. IEEE Int'l Symp. Circuits and Systems*, Vol. 1, pp. 508-512, May 1999.
- [37] B. Sunar, C. Koc, "Mastrovito Multiplier for All Trinomials", *IEEE Trans. Computers*, Vol. 48, No. 5, pp. 522-527, May 1999
- [38] H. Wu, "Bit-Parallel Polynomial Basis Multiplier for New Classes of Finite Fields", *IEEE Trans. Computers*, Vol. 57, No. 8, pp. 1023-1031, Aug 2008
- [39] H.Wu, M.A. Hasan, I.F. Blake, "New low-complexity bit-parallel finite field multipliers using weakly dual bases", *IEEE Trans. Computers*, Vol.51, No. 11, pp. 1223-1234, Nov. 2002.
- [40] H. Wu, M.A. Hasan, I.F. Blake, and S. Gao, "Finite Field Multiplier Using Redundant Representation," *IEEE Trans. Computers*, Vol. 51, No. 11, pp. 1306-1316, Nov. 2002.
- [41] T. Zhang, K. Parhi, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials", *IEEE Trans. Computers*, Vol. 50, No. 7, pp. 734-749, Jul. 2001.

VITA AUCTORIS

NAME: Mohammadali Sharifan
PLACE OF BIRTH: Tehran, Iran
YEAR OF BIRTH: 1982
EDUCATION: Allameh Helli High School, Tehran, Iran
1997-2001
Shahid Beheshti University, Tehran, Iran
2001-2005 B.Sc.
University of Windsor, Windsor, Ontario
2007-2009 M.A.Sc.